Embedded Control of Scaled Cooling System

Thesis

Presented to the Faculty of the Graduate School

Of Cornell University

In Partial Fulfillment of the Requirements for the Degree of

Masters of Science of Mechanical Engineering

By

Fisal Ali Sayed

August 2012

ABSTRACT

Buildings operations account for 41.2 percent of the total energy consumption and greenhouse gas emissions in the U.S [39]. A Large percentage of the energy consumed is wasted due to the high inefficiency of buildings. There are two conventional methods for testing energy efficient technologies in a new building: full scale prototype and computer simulation. Full scale prototypes yield actual results of a building's energy usage. However, this method is extremely expensive and resource consuming. Moreover, it is nearly impossible to compare two different designs due to the difficulties related to reproducing the building and the weather testing conditions. Pure computer simulation is also limited in its accuracy due to constraints on processing power and simplifying assumptions [21].

The overall objective of this research is to design, create and build a scaled test bed that allows engineers to better understand the behavior of buildings and contrast and calibrate computer models with real data collected from the test bed. The test bed consists of a modular and modifiable building envelope, a weather simulation enclosure and a wirelessly controllable Heating, Ventilation and Air Conditioning (HVAC) system. The thesis contributions to the overall objectives are the design and implementation of the hardware and software algorithm for the controller of the scaled HVAC system. This includes implementing Single-Input-Single-Output (SISO) PID loops to control the individual thermal states of the HVAC. A Wireless Sensor Actuator Network

(WSAN) based on Zigbee protocol is designed and implemented to facilitate two way communications between the PC and the scaled HVAC system.

Finally, an equation-based model of a centrifugal pump is created and calibrated using the pump's datasheet. A model based optimization is implemented on a centrifugal pump to maximize its efficiency.

BIOGRAPHICAL SKETCH

Fisal Sayed was born on Feb 23, 1987 in Zliten, Libya. Fisal graduated from Petroleum Training and Qualifying Institute (PTQI), Tripoli, Libya in August 2005. He was nominated by the Instrumentation and Control department at PTQI to receive a full scholarship to pursue his undergraduate studies. He received his B.S in Electrical & Electronic Engineering from University of Derby, UK in 2008.

In 2010, Fisal received a Fulbright grant to pursue his Master of Science degree in Mechanical Engineering Minor in Systems at Cornell University. In the fall 2010, Fisal began his Masters studies at Cornell University. Under the direction of professor Hencey, his studies focused on energy efficiency in Buildings, Design and implementation of controllers, and establishment of a wireless sensor actuator network. Fisal's future concentration will be on energy harvesting and sustainability and the implementation of solar power generation in Libya.

ACKNOWLEDGEMENTS

This thesis covers the work I have carried out towards fulfillment of my masters of science in Mechanical Engineering, Cornell University, 2012. I would like to express my deepest gratitude and appreciation to my advisor professor Brandon Hencey, his skilled guidance, valuable comments, stimulating discussions and support throughout the research has been indispensable. Professor Hencey's enthusiasm about the research and the practicality of work is what kept me motivated. I also would like to thank professor Garcia for his collaboration.

I would like to take this space to thank my lab mates Peter Radicki, Justin Dobbs and Nirav Patels for their valuable inputs and support all times. I owe my utmost and lifelong gratitude to my parents for their continuous backing and encouragement through my studying. My gratitude and love to my wife for her support and care throughout my studies at Cornell.

TABLE OF CONTENTS

TABLE OF FIGURES

NOMENCLATURE

Subscriptions	**Description**
HVAC	Heating, Ventilation and Air conditioning
WSAN	Wireless Sensor Actuator Network
CHWS	Chilled water supplied from the chillers
CHWR	Chilled water returning to the chillers
CHWL	Chilled water loop
PWM	Pulse Width Modulation
SISO	Single-Input-Single-Output
PID	Proportional, integral, Derivative
A	Area [m2]
Cp	Heat capacity [J/kg-K]
E	error signal
I	Current [A]
K	Thermal conductivity [W/K]
L	length [m]
m	mass flow rate [kg/s]
P	pressure [N/m2]
Q	Heat flow [W]
R	Thermal resistance
T	Temperature [K]
V	voltage [V]
W	width [m]
η	Efficiency
S	Peltier coefficient [W/A]
AS	Air Supply
Q	Heat flow transferred from one medium to another [W]

CHAPTER 1: OVERVIEW

1.1 INTRODUCTION

The main contribution of this work is the integration of a scaled HVAC system with software and hardware for the monitoring and control of the system. The first part of the work included the assembly of the scaled HVAC to control the temperature inside a scaled building. The second part of this thesis covers the design and implementation of the software algorithms and the hardware circuits to operate the HVAC system. This includes designing signal conditioning circuits for temperature sensors, actuators driver circuit, and a microcontroller support board.

Multiple PID loops are designed and implemented on a microcontroller to control the thermal states of both the chilled water loop (CHWL) and the condenser loop of the HVAC. Controlling these thermal states allows maximizing the efficiency of the performance of the thermodynamic system. A wireless sensor actuator network (WSAN) is implemented via the Zigbee protocol to collect data from the system and store it in MATLAB for further analysis, and online calibration of HVAC components. The WSAN protocol also allows the remote tuning of the microcontroller PID parameters, set-point, etc.

1.2 PROJECT MOTIVATIONS

HVAC systems are pervasive energy consumers (Figure 1-1). More interestingly, a large percentage of the energy consumed is wasted due to the highly inefficient buildings. Therefore, there is still much room for improvement in the efficiency of HVAC controllers

and making buildings more energy efficient and more self-aware with technology that already exists. Buildings computer models are limited by their assumptions and do not actually represent the real buildings because they do not perform as expected. "*Building simulation experts are wary to suggest that modeled energy use equates to actual energy use" [34].* The overall research aims at building a complete test bed that will facilitate running experiments to validate simulation assumptions. Buildings are subject to constantly changing conditions. Therefore, it is challenging to recreate experimental conditions for the purpose of comparison and verification. This test bed will allow experiment repeatability and permit prototyping of new hardware technologies.

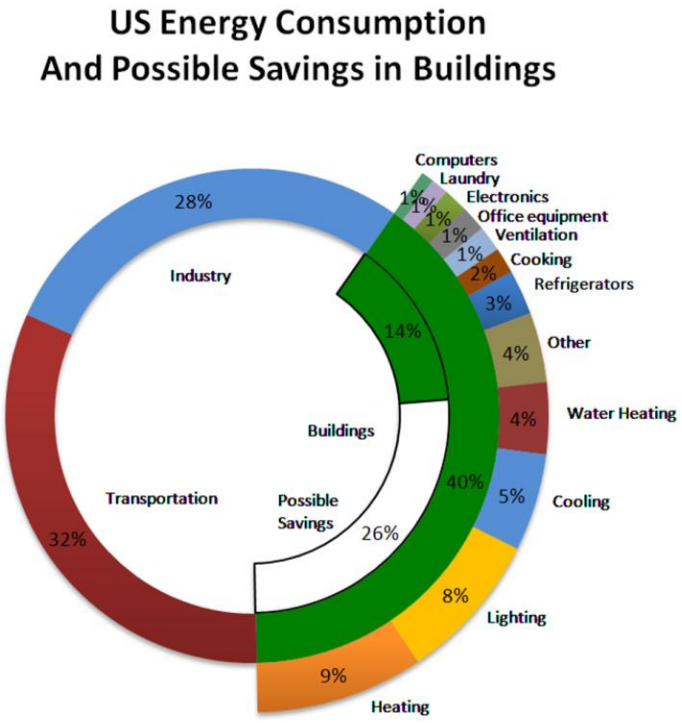


Figure 1- 1: Energy consumption distribution in the U.S. [13]

The scaled HVAC system in particular will enhance understanding of building dynamics and their controllers. The nature of a physical model is different from that of a simulation, with a real model having strengths and weaknesses that complement those of simulations. The applications of the work and the practical experience that I gained during the implementation phase of this project are invaluable.

1.3 SCOPE OF THE THESIS

This thesis's goals were set to facilitate accomplishment of the primary goals of this research project. The following lists the primary goals of the research and the sub-goals of this thesis separately:

Primary goals of the Research

The overall goal of this research is to create a scaled test bed that will validate buildings simulation assumptions, cheap-rapid prototyping of new hardware technologies and advanced controllers implementation. The primary goals of this research are as follows:

- Design and build a modular and easily modifiable scaled building envelope.
- Design and build a weather simulation enclosure that will simulate external weather such as solar radiation, external temperature, wind etc.
- Build, assemble and fully control a modular scaled HVAC system to control the weather inside the scaled building.
- Design and implement a wireless sensor actuator network for collecting data about the scaled HVAC, the exterior and interior climate of the scaled building.

Thesis project goals:

This thesis contributions were set to accomplish the last two goals listed above. To accomplish these two goals, the following sub-goals are accomplished:

1. Design and build signal conditioning circuits and actuators driver circuit to interface sensors and actuators with the Mega 328 microcontroller.
2. Programming the microcontroller for set-point tracking and fault management.
3. Design and implement a wireless sensor actuator network to transmit and receive analog and digital data between the HVAC system and a PC.
4. Demonstration of online optimization to maximize centrifugal pump efficiency.

1.4 OUTLINE OF THE THESIS

The rest of this chapter provides an explanation of the scaled HVAC system setup and its general design requirements and principle of operation. The rest of this thesis is organized into the following chapters: chapter 2 provides a review of PID controllers and their tuning process with emphases on designing and tuning multiple SISO control loops.

Chapter 3 discusses the detailed design and implementation of the hardware and software of the HVAC controllers. This includes a review of microcontroller's hardware peripherals that is necessary for the project. A detailed electronic circuit board design is also outlined. An explanation of the design and implementation of WSAN and the GUI is also included. Chapter 4 presents the work done on modeling a centrifugal pump as a first step towards modeling the whole HVAC system. The pump model uses polynomial curve fits in order to find the parameters of the efficiency equation and the head equation. A

model based optimization is implemented on the centrifugal pump model to maximize its efficiency.

Chapter 5 summarizes thesis results and conclusions drawn from the research. Future recommendations are given to improve the current HVAC performance as are some solutions to problems faced during the testing process. Appendix A contains the parts list and costs. Appendix B contains the commented Arduino controller code. Appendix C contains a commented WSAN MATLAB GUI code. Appendix D reports the Modelica pump model code. Appendix E explains the troubleshooting process needed for the GUI.

1.5 SCALED HVAC SYSTEM REQUIREMENTS

Table 1-1 summarized the general requirements of the scaled HVAC system which the final setup was based on.

Part	General Requirements Description
Temperature sensor	• It shall be ±2% accurate. • It shall have a fast response time. • It shall use a standard 2 wire connection. • It shall be small enough to be immersed in ½" piping. • It shall have a range of at least -10°C to 125°C.
Physical scaled HVAC system	• It shall be easy to reconfigure via modular design. • The system shall be easy to connect and disconnect from the building envelope. • It shall use standard connectors and piping size. • It shall be easy to access its components.
Electrical & electronic requirements	• The hardware shall be safe to use and shall include over-current protection. • There shall be a use of ground shields and thick ground planes. • It shall use preventive safety measures in power and output wire connectors. • The low voltage and high voltage components shall be decoupled. • High current components shall have minimal impact on the low voltage components. • Electronic circuits shall be modular in design and easy to debug.

Software Algorithm	• It shall be implemented with minimum use of floating points. • It shall protect the Peltier module and the pump from overheating. • It shall have accurate time base. • Data shall be transmitted and received via Zigbee protocol. • It shall have a GUI to facilitate communication and controller parameter tuning.

Table 1- 1: General requirements of the scaled HVAC system

1.5.1 Modularity Design

One of the main requirements of the scaled HVAC system is modularity of design in order to maintain a dynamic, versatile apparatus [21]. Modular design facilitates any future changes in the system setup. Each individual sub-circuit is designed and built on a separate circuit board and mounted on a single main board. Each board has its own power connector and output connectors. This modular design will facilitate the PCB manufacturing process in the future and reduce the cost of manufacturing. Furthermore, the hardware circuits can be easily expanded by stacking new boards on the existing electronic circuits. Figure (1-2) illustrates the high level structure of the test bed.

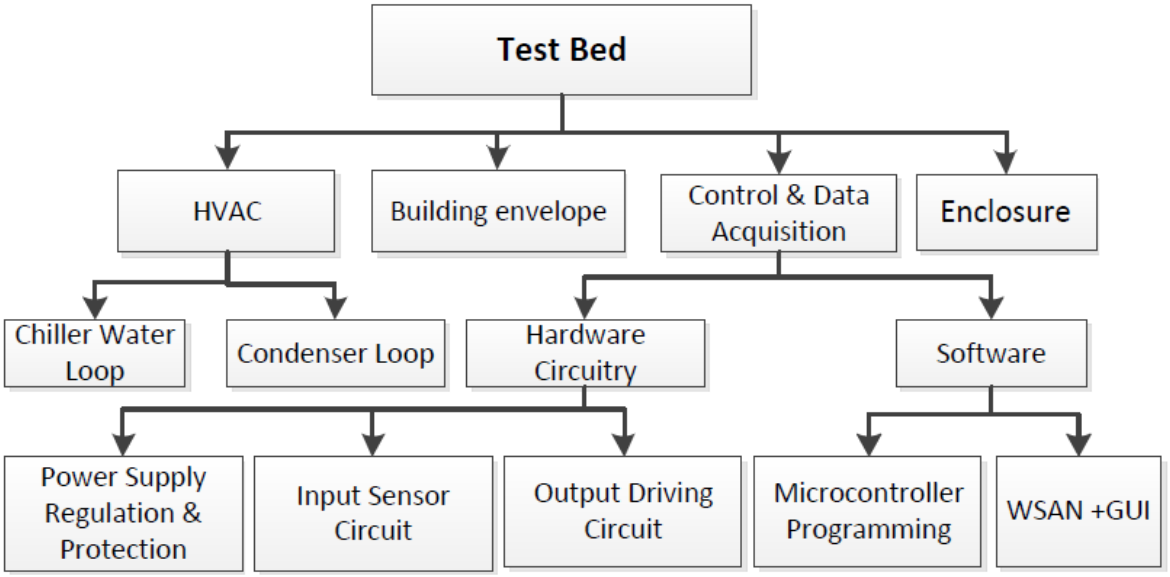


Figure 1- 2: High level design of HVAC system

1.6 HVAC CONTROL SYSTEM ARCHITECTURE

HVAC systems mainly consist of two loops: a chilled water loop CHWL and a condenser loop. The CHWL is usually comprised of a cooling tower and a chiller which work to cool the chilled water return (CHWR) and provide a chilled water supply (CHWS) at a predetermined flow rate and temperature [14]. The cool air supply absorbs heat in the conditioned space, and the warmer air returns through the air duct back to the air handling unit (AHU). Throughout the building, AHUs use the chilled water supply to cool down the returned air and provide the necessary ventilation using outside air (Figure 1-3). The cooled air supply leaves the cooling coil and the air cycle repeats. At the zone level, a damper is used to modulate the air supply [14]. In the heating/cooling plant, there is a condenser loop which basically removes heat from the CHWR by allowing the CHWR to flow into the refrigerant tubes [14].

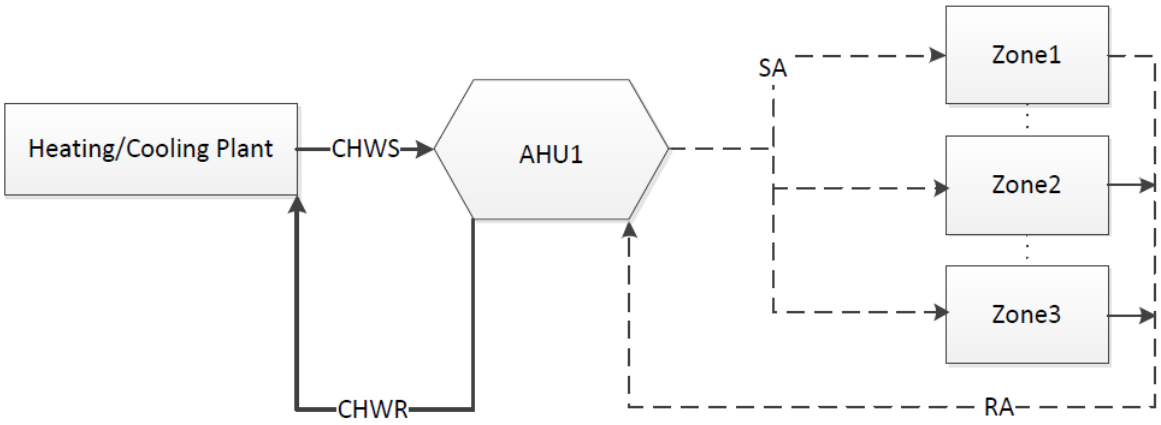


Figure 1- 3: Typical hydraulic cooling system

Figures 1-4, 1-5 show the condenser loop side and the CHWL side of the scaled HAVC respectively.

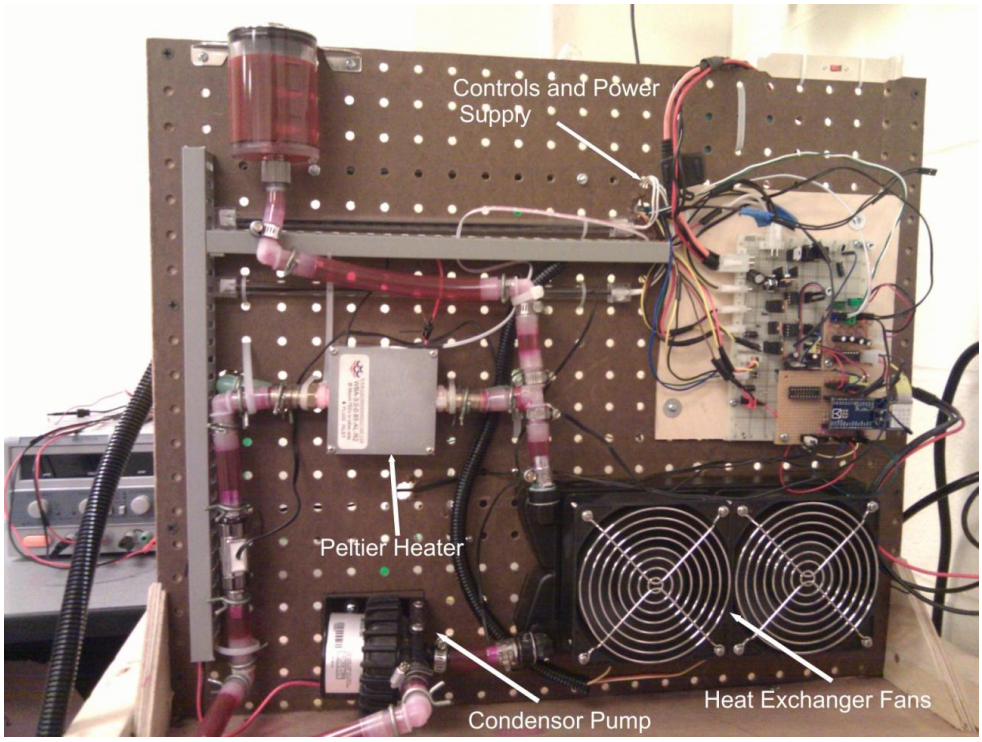


Figure 1- 4: Condenser loop of the scaled HVAC system

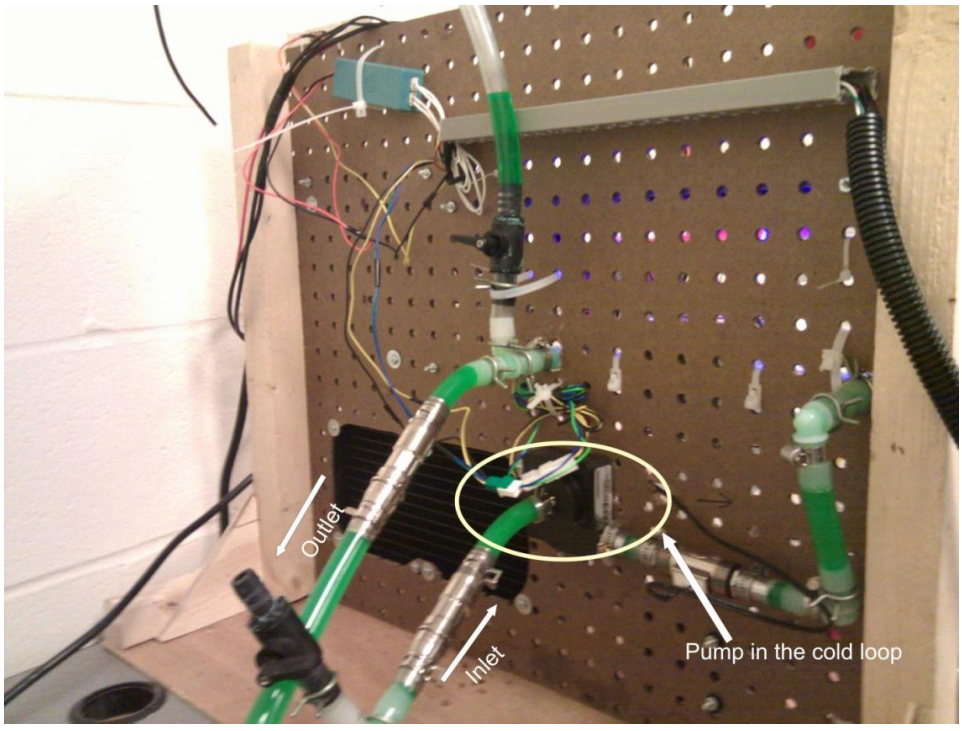


Figure 1- 5: CHWL of the scaled HVAC system

1.7 SCALED HVAC SYSTEM SETUP

Team members met and decided on what the scaled HVAC should consist of in order to mimic the real HVAC system. Previous students performed basic calculations to select the correct size of each component. The final scaled HVAC system consisted of two main loops: the CHWL and the condenser loop as shown in Figures 1-4, 1-5. The chilled water loop consists of a water block attached to the cold side of the Peltier module. The coolant flowing through the water block acts as a heat source on the cold side of the Peltier module [2]. The CHW supplied from the water block goes through a cooling coil in the conditioned space to absorb heat. The hot water returning from the cooling coil is circulated by a centrifugal pump to go through the water block (chiller). This process then repeats itself as indicated in (Figure 1-6).

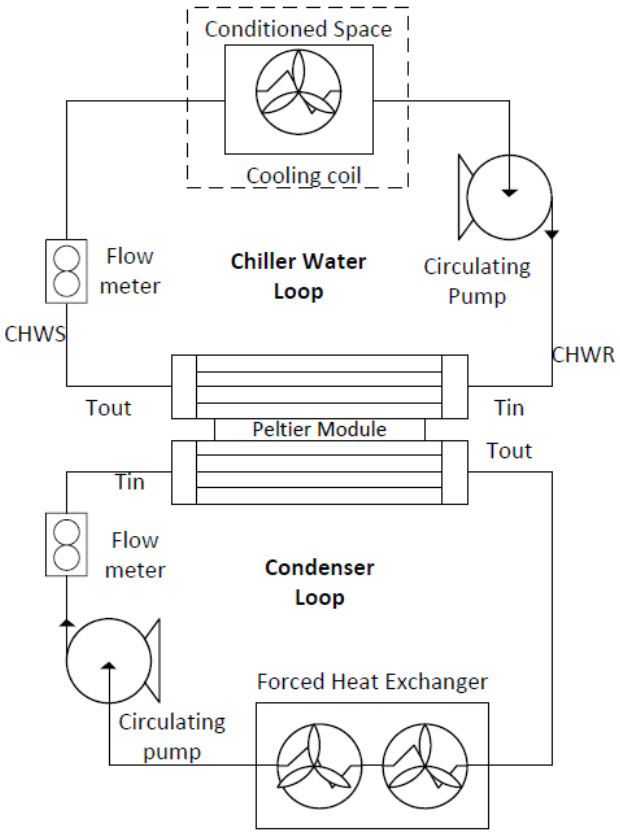


Figure 1- 6: Schematic diagram of the two loops of the scaled HVAC system

The condenser loop is made of a water block attached to the hot side of the Peltier module. Water acts as a heat sink on the hot side of the Peltier module. Heat is removed from the chiller water loop and added to the condenser loop by the Peltier Effect [18]. The more heat removed from the condenser loop, the cooler the CHWL temperature becomes; thus, the more heat taken out of the conditioned space.

CHAPTER 2: DESIGN AND IMPLEMENTATION OF PID CONTROL LOOPS

2.1 INTRODUCTION

To better control the thermodynamic cycle of the HVAC, it is desirable to control the individual thermodynamic states of the chilled water loop and the condenser loop with an inner loop, supervisory algorithms can be used to maximize the coefficient of performance as shown in Figure 2-1 [35]. In practice, it is more desirable to tune multiple SISO control loops rather than a single MIMO control loop. However, the condenser and the chilled loops are both closed cycles, implying strongly coupled system dynamics [35]. It has been shown that multivariable control techniques [36]-[37] can be used to handle input-output coupling while achieving desired performance objectives. In this thesis, SISO control loop is implemented where the system parameter is decoupled from the rest of the system, whereas MIMO can be used to control the system dynamics that are highly coupled.

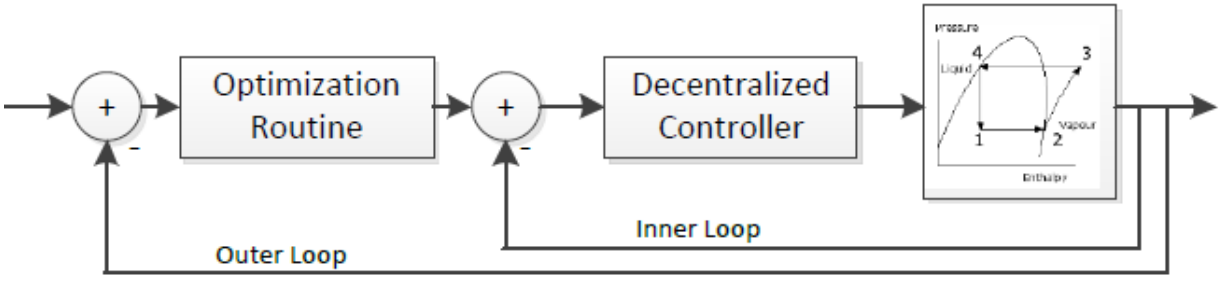


Figure 2- 1: Block diagram visualization of an inner decentralized controller with a larger optimization control loop.

2.2 PROPORTIONAL, INTEGRAL, DERIVATIVE PID CONTROLLER REVIEW

PID controller calculates its output based on the measured error (e) and the three controller gain parameters; the proportional gain (K_p), the integral gain (K_i) and the derivative gain (K_d). The proportional gain is simply multiplying the error by a gain factor K_p. The error decreases with increasing K_p but the system becomes oscillatory if K_p is too large, on the other hand, if K_p is too small, the system becomes sluggish and may have a steady state error. The integral control is added to automate readjustment of control signal for the new equilibrium position; hence the integral control provides better steady state accuracy. The derivative term improves stability of the closed loop system by minimizing the overshoot; however, in most of the HVAC systems, it is not necessary to use the derivative part, because it tends to amplify high frequency noise. Hence only proportional and integral parts of the PID controller were implemented. Figure 2-2 shows the structure of a parallel PID controller. The output of the controller U is the sum of these three terms:

$$U = K_p * e(t) + K_i \int e\, dt + K_d \frac{de}{dt}$$

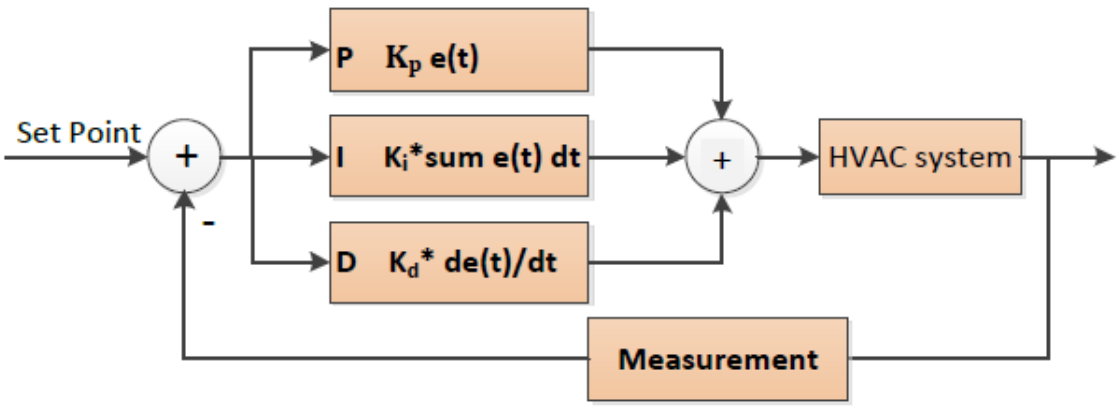


Figure 2- 2: Parallel architecture of PID controller

Table 2-1summarizes the advantages and disadvantages of using different controllers

Controller	Pros	Cons
P	• Easy to implement	• Long settling time • Steady state error
PD	• Easy to stabilize • Faster response than just P controller	• Can amplify high frequency noise
PI	• No steady state error	• Narrower range of stability

Table 2- 1: Summary of the effect of PID terms on a system

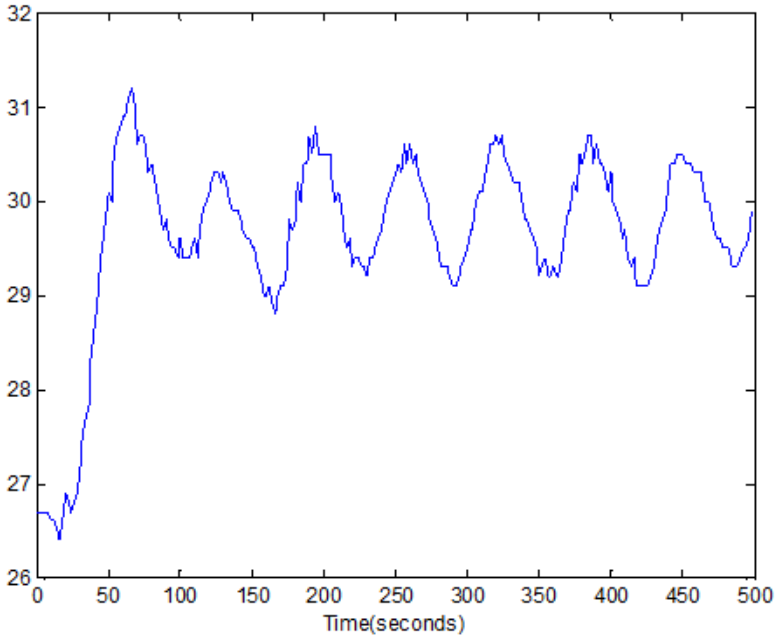
2.2.1 Saturation and anti-windup

The wind up action happens when the output of the controller U reaches saturation (≥ 255 for 8bit PWM signal). Since error is still positive, the integral part continues to increase causing more inputs applied to the system. This causes the control signal to remain in saturation and the feedback loop is basically broken. This windup action is avoided by restricting the output signal of the controller u to be within the acceptable range of the actuator. Here, the output is restricted between 0 - 255 (8bit PWM signal) as shown in the following pseudo-code:

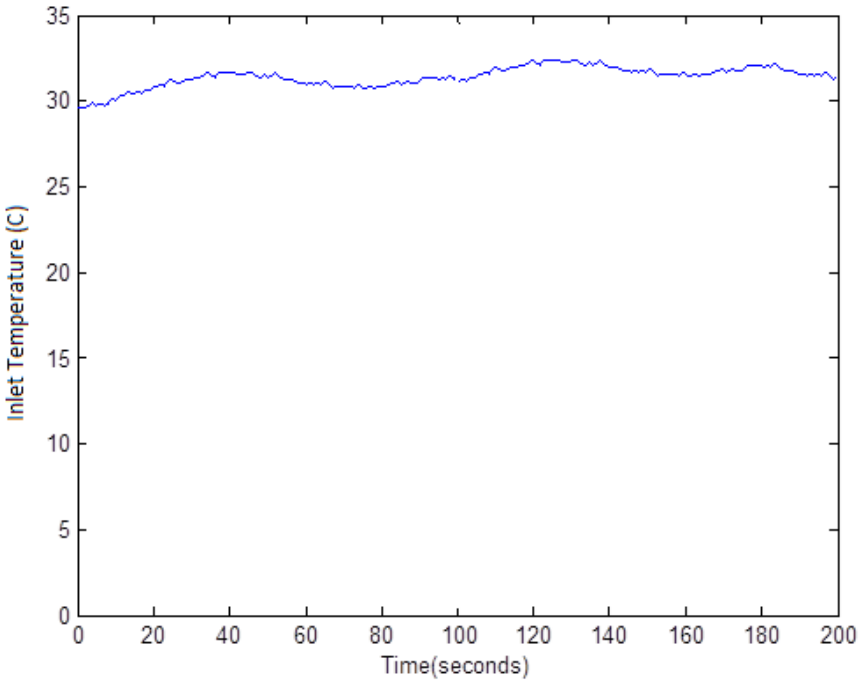
```
if (u > p->max)  // p is a structure
{
  {u = p->max; /* Clamp the output */ max=255
  if (error > 0) /* Error is the same sign? Inhibit integration */
    {int_ok = false;    }  }
  Else  if (u < p->min)    /* Repeat for negative sign */
    {    u = p->min;      // min=0
  if (error < 0)
    {  int_ok = false;    }  }
  if (int_ok) /* Update the integrator if allowed. */
    { p->i = new_i;  }
return (int)u;
}
```

2.2.2 PID Controller Tuning Parameters

There are a couple of strategies on how PID controllers can be tuned; this includes trial and error tuning method which is based on the experience of the user or Ziegler-Nichols tuning method which is based on the step response of the system [29]. The approach followed here is calculating the controller output function U such that the final output signal (U) applied to the driver circuit was divided by the number of shifts determined in PI structure. This is a systematic way of changing the PI parameters. I started by 7 shifts to the right; meaning that the output signal is divided by $2^{\wedge 7}$ or the value 128 in decimal. This way we are half way through either increasing division factor by 128 or reducing it by 128. Increasing the division factor reduces the effect of changing the PI gain parameters. I started with only applying a proportional control by setting $K_i=0$, doubling the value of K_p until a constant amplitude oscillation was obtained (Figure 2-3a). To improve on the effect of K_p, an additional K_i was set and K_p value was reduced to about 60% of its value. Adding integral gain had a great effect on reducing steady state error between the desired temperature and the actual temperature (Figure 2-3b). The final PI controller gave a stable and fast response to meet the set point after the PI controller was properly tuned.



a) Proportional controller response on SISO



b) PI controller response after proper tuning for SISO.

Figure 2- 3: PID parameter tuning

2.2.3 Two SISO PI loops sharing the same input

There are two degrees of freedom that affect achieving the thermal state of the condenser inlet heat exchanger temperature; the air flow rate through the heat exchanger which is controlled by the fans speed and the voltage applied to the Peltier module which controls how much heat is conveyed from the chilled loop to the condenser loop. The objective of this control loop is to control the two actuators to meet a desired temperature for the inlet of the heat exchanger. Figure 2-4 shows two PI loops aiming to meet the set-point for the inlet of the heat exchanger. The advantage of such a method is that the settling time and the amount of oscillation is drastically reduced (Figure 2-5). During the tuning process, it was noticed that the integral control of each loop is causing instability to the other loop. This problem is solved by only applying a P-control on the Peltier module and a PI-control on the fans. The rest of the tuning process is exactly the same as the tuning process for SISO loop. Figure 2-5 plots the temperature transient response of the heat exchanger inlet temperature after tuning the two loops.

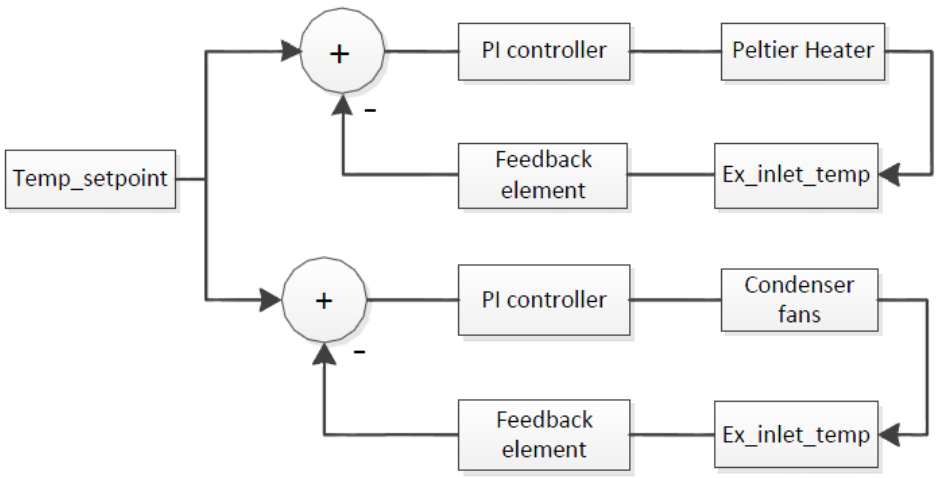


Figure 2- 4: Two SIMO PI loops

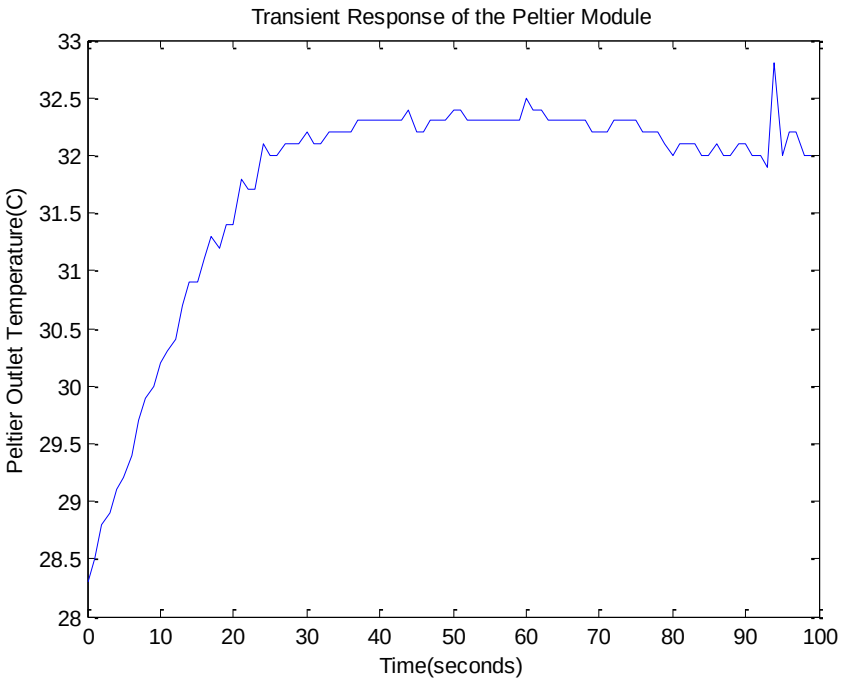


Figure 2- 5: Temperature step change response

When the desired set-point is achieved, the Peltier power consumption starts to reduce until it settles down to about 150W as shown in Figure 2-6.

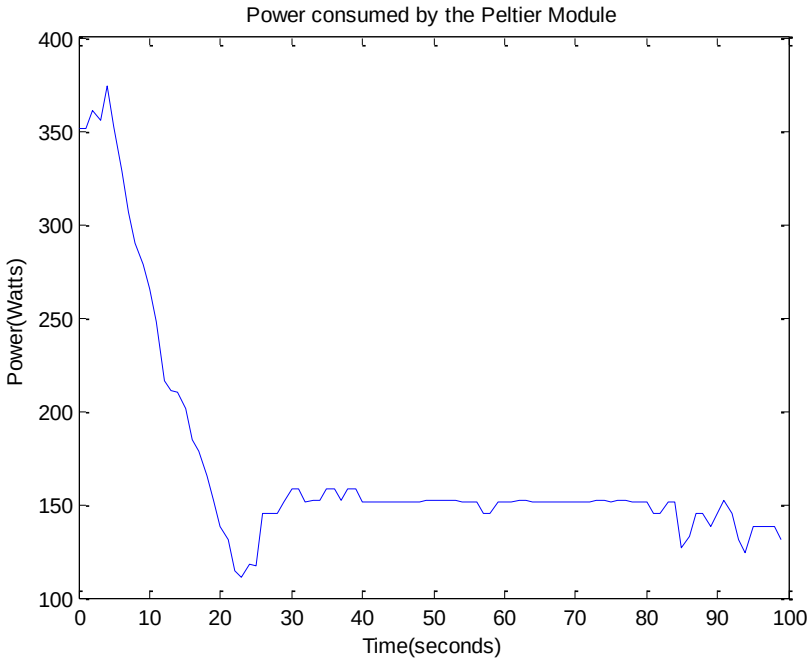


Figure 2- 6: Peltier module power consumption for the set point of 32.3C

2.2.4 Multiple SISO PID loops

Multiple SISO PI control loops are designed to control different thermal system parameters. Three SISO PI control loops were implemented (Figure 2-7). A PI control loop (K_3) is implemented to control the CHWS temperature, the other two PI control loops (K_1, K_2) aimed to control the temperature difference across the heat exchanger. Due to the small variation in chiller loop temperature, K_3 switched the CHWL pump on all times to achieve T_CHWS set-point. This loop is decoupled from the other two control loops and does not affect their performance.

The temperature difference across the heat exchanger is affected by fans speed and Peltier module. The Peltier module acts to achieve the desired inlet temperature whereas the fans cool down the outlet temperature to a desired temperature. It is clearly shown that these two loops are highly coupled and act as disturbance to each other. Their tuning process is a challenging task since both loops are highly coupled.

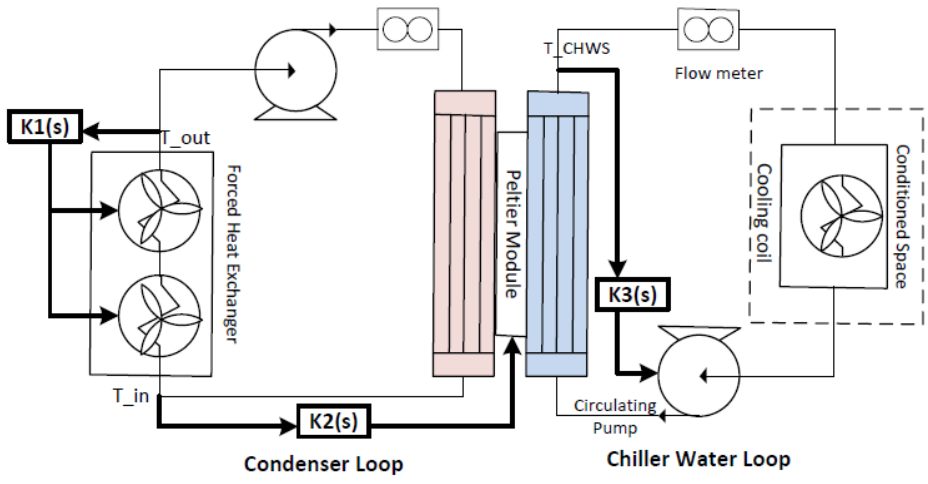


Figure 2- 7: Three SISO PI control loops

Due to actuator limitations, not all combination of set points can be tracked. Finding a reasonable set point was achieved by letting the system reach a steady-state in open-loop while providing 70% PWM signal to each actuator. Tuning these two loops was achieved by setting the K_i to minimal value while increasing the value of K_p. The fan PWM signal was limited to maximum of 70% to constrain its impact on the temperature response. After establishing a basic intuition, best performance was achieved within ±0.5°C of the set-point as shown in Figure 2-8.

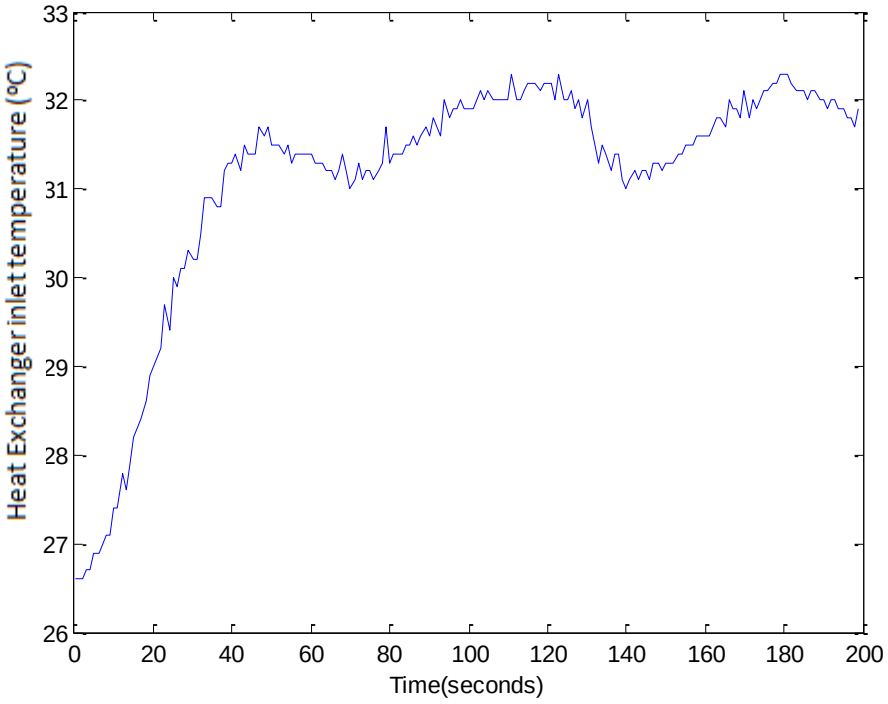


Figure 2- 8: Temperature step change response.

CHAPTER 3: CONTROLLER DESIGN & IMPLEMETNATION OF WSAN

3.1 INTRODUCTION

This chapter presents the work done on the software algorithms and the electronic circuits design. It discusses the functional requirements of the controller hardware choices and the integral decisions made to select suitable controller hardware for the system. A detailed design of sensor signal conditioning, output drivers, and a microcontroller support circuits is presented. This chapter also provides the design and implementation of the wireless sensor actuator network (WSAN) and the Graphical User Interface (GUI) designed to facilitate communication with the HVAC system.

3.2 HARDWARE CONTROLLER SELECTION

The main functional requirements and specifications of the hardware controller are briefly listed in Table 3-1.

Requirements	Description
Controller Hardware	• It shall have as many ADC channels as necessary. • It shall be capable of generating at least 6 PWM signals to control all the actuators. • It shall have a predictable software execution time. • It shall have timers to provide a stable periodic interrupt. • It shall be able to interact with a wireless network. • It shall be able to measure flow meter frequency. • It shall be expandable if more analog or digital channels are required. • It shall use a standard communication protocol.

Table 3-1: Functional requirements of the scaled HVAC controller hardware

A Number of control hardware solutions were considered, including LABVIEW Software with NI DAQ cards, MATLAB Software with xPC target, and microcontroller-embedded system. LABVIEW software was not selected because it is hard to debug and is not well documented. In addition, the high cost and limited scalability of NI cards suggested that proprietary data acquisition cards were not the best choice. MATLAB requires xPC target to compile the Simulink model into C code and runs using a real time operating system on a dedicated computer, and it also requires NI cards.

In contrast to PC-based data acquisition cards, microcontrollers are cheaper, and more scalable; if needed, controllers can be combined to yield as many PWM signals as necessary. Thus, the project team selected the Arduino Fio board which is based on the Atemga 328p microcontroller. This board has all the essential features required, some of which will be discussed in the following sections. Table 3-2 illustrates the basic criteria that the hardware controllers were compared against.

Criteria	**Portability**	**Level of support**	**Cost**	**Ease of use**	**Wireless network capability**	**# of counters**	**Total score**
LABVIEW software & NI DAQ	4/10	3/10	3/10	2/10	5/10	2/10	19/60
MATLAB + xPC target + NI DAQ	4/10	7/10	3/10	7/10	5/10	2/10	28/60
Microcontrollers	10/10	9/10	10/10	9/10	10/10	10/10	57/60
Weight	3	2	5	4	4	3	

Table 3- 2: Controllers hardware decision matrix

3.2.1 Analog to Digital Converter Setup

To process analog signals in a microcontroller, an analog to digital converter (ADC) is needed. Generally, all external sensors output analog signals such as temperature sensors and current sensors. Arduino Fio board has 8 analog channels (A0-A7) based on a 10 bit ADC. To set up the ADC, the following two registers are necessary:

- ADMUX: This register sets the reference voltage for the ADC and selects the appropriate channel.
- ADSCRA: This register has control over the enabling and disabling of the ADC, starting a new conversion on a channel, enabling the ADC interrupts and selecting the clock frequency for sampling the input. Figure 3-1 illustrates a sample code to set up the ADC. For detailed bit manipulations, refer to the Analog to Digital Converter section in the Mega328p datasheet.

Sample code:

```
void adc_init(void)  // function to initialize ADC
{
  PRR &= ~0x01; /* Disable ADC power-down logic */
  ADCSRA |= ((1<<ADPS2)|(1<<ADPS1));// 125Khz the ADC   reference clock
  ADMUX = 0x40; /* Use 5V reference, start at channel 0 */
  ADCSRA |= (1<<ADEN); //Turn on ADC
  ADCSRA |= (1<<ADSC); //Do an initial conversion because this one is the
slowest and to ensure that everything is up and running
}
```

Figure 3- 1: Sample code of setting up the ADC

3.2.2 Hardware Triggers and Interrupts

An interrupt is a way for an internal or external event to pause microcontroller activity so that it completes a task in the interrupt service routine (ISR). This systematic occurrence establishes a stable time base for the controller. Thus, software counters are incremented or decremented in the ISR to keep a time base for other tasks in the main program. Data shared between the ISR and the main program is defined as volatile. This tells the compiler that the variable is shared and may be subject to outside changes. In the ISR interrupt, ADC measurements are sampled and synchronized to sample ADC channels when the pulse width modulated signal PWM is on. Synchronization of ADC channel sampling has the advantage of avoiding high frequency switching noise, so that measurements are only taken during the on period.

3.2.3 Integer versus Floating Point Calculations

Floating point arithmetic involves high precision numbers that is convenient to use. However, it does not show any more precision since Arduino Fio board has a 10 bit ADC. Converting a 10 bit value to a 32 bit float does not provide any more precision, because we are not creating new information when we scale the ADC value up. In addition, the AVR core does not have a floating point unit (FPU); it needs to perform these calculations on floats in an indirect way. Therefore, floating point calculations take longer than integer calculations. In addition, in some situations we may not be utilizing the actual capabilities of float, and this merely wastes CPU time and memory. For instance, if we define float x=3, although 3 is an integer the CPU will perform floating point operations instead of integer operations and increase the overhead on processing time. Floating point arithmetic is avoided through all the calculations. Fixed point arithmetic is used to convert the ADC

values into temperature. The user is only allowed to enter an integer value when tuning the PID controller and if a smaller value is needed, we increase the number of shifts to the right in the PI structure (see appendix B for complete code)

3.3 WIRELESS COMMUNICATION PROTOCOLS

For the wireless communication, Zigbee protocol was selected. This protocol is based on IEEE 802.15.4 Zigbee technology and is used with low power wireless radio in communication systems which require high battery life and secure networking but can tolerate only a low data rate. The Xbee transceiver, manufactured by Digi, implements this protocol and is directly compatible with the Arduino board's UART interface. Hence, communication with the PC at the receiving end is also straightforward. All the modules with the same Personal Area Network ID form a mesh network. Therefore, a master node is not required. This is the biggest advantage of using this protocol. Figure 3-2 shows a simple point to point communication between an Xbee module connected to a PC and different Xbee module connected to the Arduino Fio board.

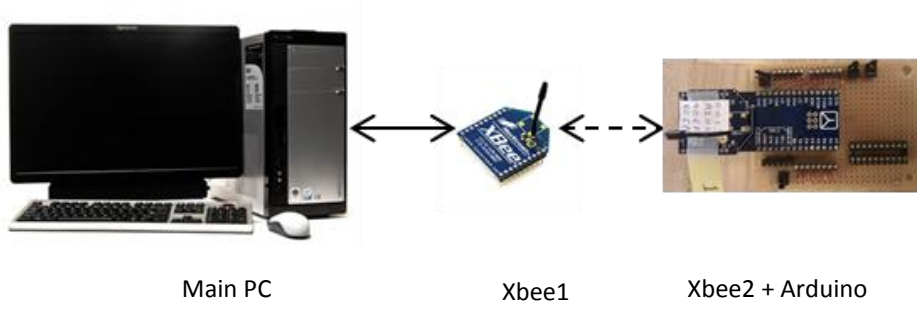


Figure 3- 2: Two way communication set-up between PC and HVAC system

3.3.1 Zigbee Protocol

Zigbee protocol is designed for digital radios which work at low power and low data rates. It transmits at a 250 kbps data rate, which is sufficient for temperature sensors whose readings change slowly. The advantage of using Zigbee is that it can automatically create an ad-hoc network amongst the nodes. The user does not have to set up any configuration; since setting the correct parameters in the module is all that is required for the node discovery and network formation. The WSAN is setup between a coordinator Xbee and a router or an end device Xbee. The coordinator coordinates activity within the network while interacts with the outside world. The end device node reports to the coordinator. In this project, the Xbee transceiver attached to the PC is configured as a coordinator and the Xbee attached to the Arduino on the HVAC side is configured as a router. X-CTU software is used to configure the Xbee transceivers [30].

3.3.2 API Wireless Protocol

The Zigbee modules work in both modes: API mode, and transparent mode. API mode provides more encapsulation for the data and is useful in setting up a mesh network to deal with data from multiple nodes. In mesh architecture, every node can be a router for its neighbor nodes. Thus, the network management and data flow is decentralized. The coordinator is only responsible for initiating the network, but it is not instrumental to the routing of traffic across the network. If the coordinator is removed or temporarily inaccessible, there is variety of other paths that the data could take to reach the destination. This kind of redundancy and decentralization makes for much more scalable network architecture with long term reliability and adaptability. The advantage of API mode is that each frame has a payload which consists of information on the senders address, number of

data bytes in the frame and checksum for error checks, but the drawback is that the decoding of data becomes complex at the receiving end. This creates a tradeoff between the complexity of the design and the features offered.

3.3.3 Transparent Protocol (AT mode)

The transparent version is a more convenient form of communications. Use of this mode is best suited for point-to-point communication. The transceiver emits data from another node on the UART. Similarly, it accepts the data from the serial terminal without having to form any framing operations manually. This makes the decoding tasks at the receiver simple and straightforward. Nevertheless, in this mode, the Xbee transceivers communicate within them using the Zigbee protocol now hidden from the user. The drawback here is that the module truncates all the frame information provided in the API mode, and instead just puts out the actual data transmitted on the serial. This simplifies point-to-point communication but hides the sender information from the user. So, to implement a mesh network, we need to send out the address of the sending node before transmitting the actual data.

Using the transparent mode will help students who will continue this research project to make changes quite smoothly compared to the API mode. As a result, the AT mode is used keeping in mind the future developments necessary for the system.

3.4 HARDWARE CIRCUIT DESIGN

The hardware circuit design was divided into three main sub circuits (Figure3-3): temperature sensor signal conditioning, output drivers, and the microcontroller support circuit. Each sub-circuit is implemented on a separate board. The output driver board contains the voltage regulators and current sensor circuitry. In the following sections, each of the three sub-circuits is briefly discussed.

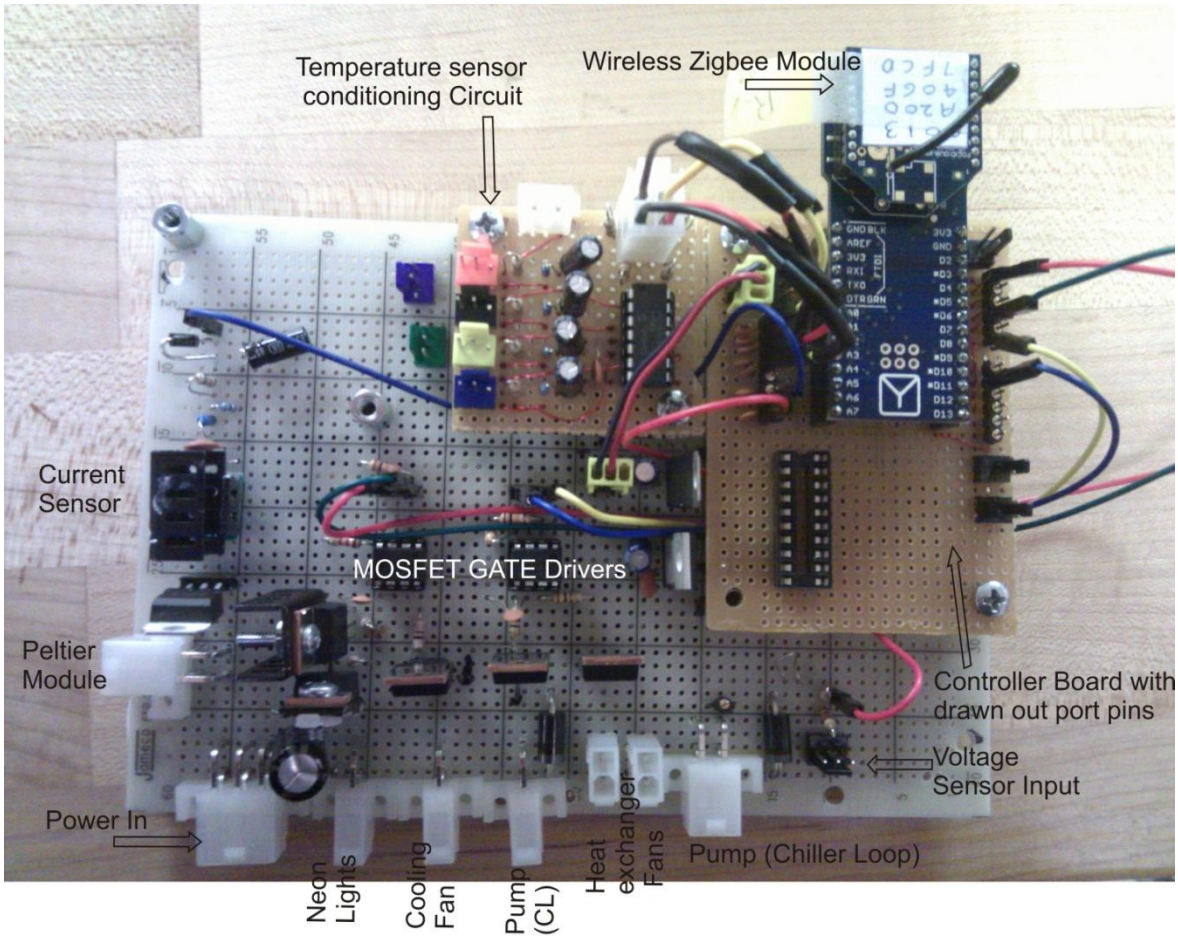


Figure 3- 3: Final hardware design board

Table 3-3 summarizes the hardware connection of each input sensor and output actuator with the Arduino controller board. It also lists the voltage supply required by each actuator and sensor.

Connected Actuator/Sensor	**Pin on the Arduino Board**	**Pin on the Controller**	**Voltage supply required**
Peltier Heater	D11	PORTB.3	24V
Pump in the condenser loop	D4	PORTD.4	12V
Heat Exchanger Fans (condenser loop)	D9	PORTB.1	12V
Pump in the chiller loop	D2 or D4		12V
Fan in the chiller loop	D3	PORTD.3	12V
Peltier Outlet Temperature	A3	PORTA.3	3.3V
Heat Exchanger (condenser loop) outlet Temperature	A4	PORTA.4	3.3V
Peltier Hot Side Temperature	A1	PORTA.1	3.3V
Fan Out temperature (cold loop)	A0	PORTA.0	3.3V
Current Sensor	A6	PORTA.6	5V
Voltage Sensor	A7	PORTA.7	NA
Flow Sensor (hot loop)	D12	PORTB.4	3.3V
Flow Sensor (cold loop)	D13	PORTB.5	3.3V

Table 3- 3: Hardware pin connection with Arduino board

3.4.1 Temperature Sensor Circuit:

10k negative temperature coefficient Thermistors are used in the system. A signal conditioning circuit (Figure 3-4) consisting of a second series resistor and buffer amplifier converts resistance into a low-impedance voltage that can be read by the ADC. The pull-up resistor value of 4.7KΩ yields maximum resolution at the temperature range of interest (Figure 3-7). The buffer prevents the ADC from loading the Thermistor while providing the low impedance necessary to rapidly charge the sample-and-hold capacitor in the ADC during high speed operations. MCP6004-I/P Quad Op Amp was chosen because it has very high input impedance and has four op-amps on one chip. A look-up table in the microcontroller converted the nonlinear temperature-voltage characteristic of the circuit into linear Celsius-scaled integer units. Thermistors were calibrated using an accurate standard thermocouple sensor by putting Thermistors in a cup containing boiling water and as it cools down, temperature and resistance value of Thermistors are recorded.

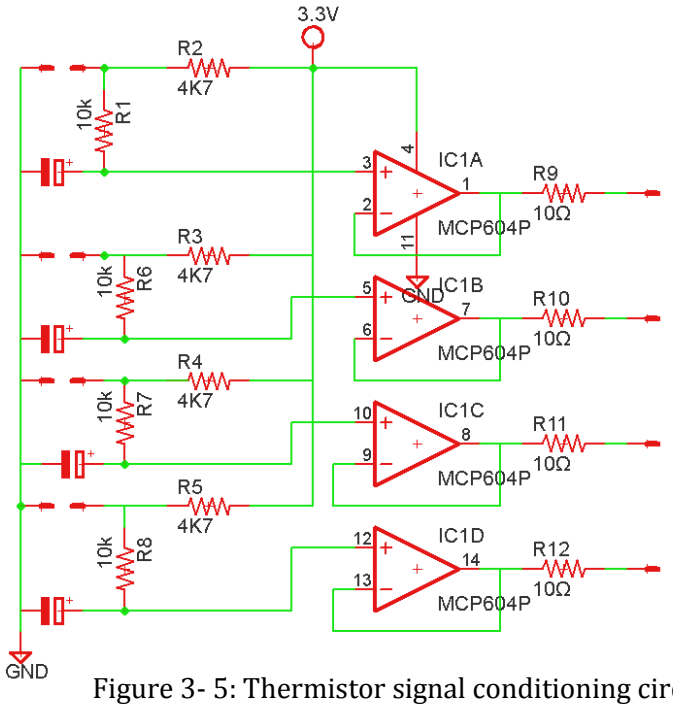


Figure 3- 5: Thermistor signal conditioning circuit

Figure3-6 shows the use of the ADC full range over the temperature range from -10 to 90°C. Secondly, because a voltage divider requires fewer components than a Wheatstone bridge, tolerance stack up is less significant.

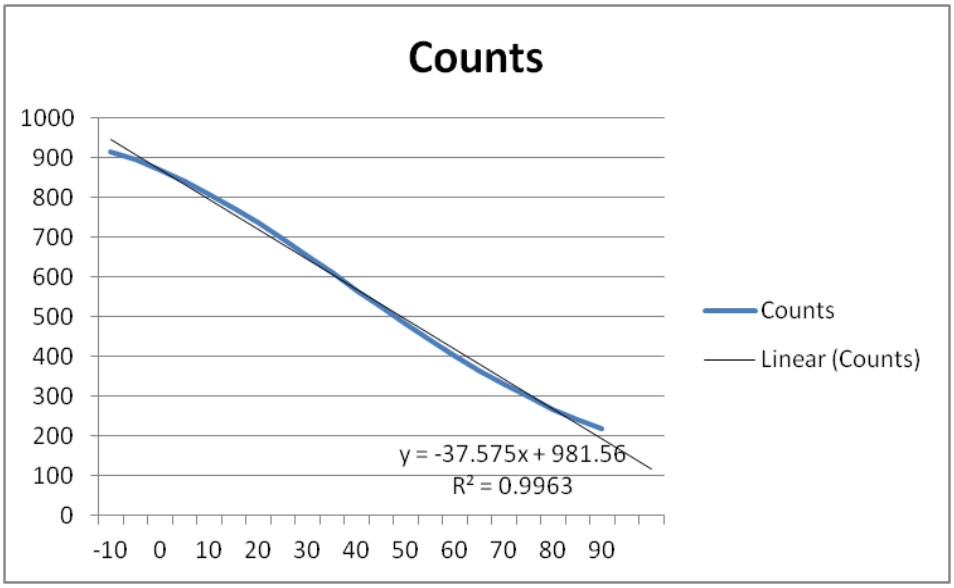


Figure 3- 6: ADC counts over the temperature range

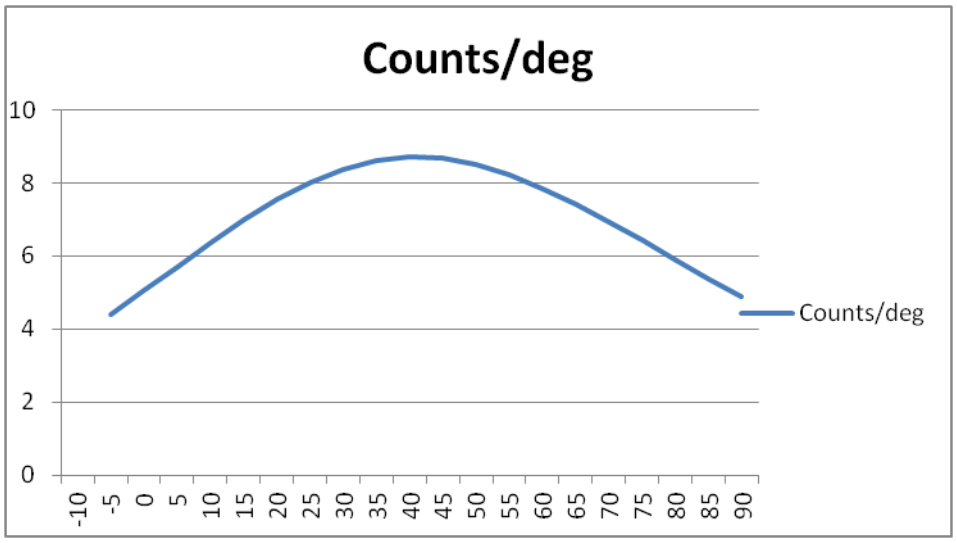


Figure 3- 7: Analog to digital converter resolution

3.4.2 Actuator Driving Circuit

There are six actuators on the scaled HVAC system that need to be controlled. These actuators are listed in Table3-4.

Actuators	Functionality
Swiftech MCP-35X pump	• Chiller water loop pump, PWM controllable.
Swiftech MCP 655 pump	• Condenser Loop manually + PWM controlled.
Peltier Element	• Remove heat from the chiller loop and conveys heat to the condenser side.
Two Condenser Loop fans	• Dissipate the heat gained in the condenser loop.
Cooling coil Fan	• Cools down the conditioned space.

Table 3- 4: Scaled HVAC system actuators

To drive these actuators, a gate driver, IXDN604PI, switches the power MOSFETs in a low-side configuration (Figure 3-8). A 10Ω resistor in series with the input of the MOSFET limits the switching speed to control noise from fast switching transients. Pull-down resistors at each gate driver input prevent signals from floating which might lead to inadvertent MOSFET activation.

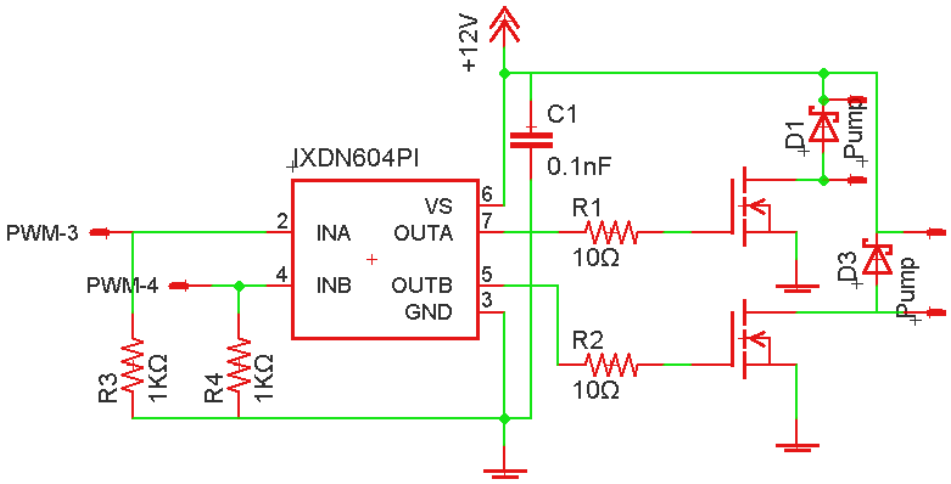


Figure 3- 8: Output driving schematic circuit

The IRLB3034 MOSFET is used to drive the actuators because it can sustain a high current through the drain ($R_{DS\,(on)}$ =1.4mΩ), dissipating about 1Watt when I_D is 28A. Two MOSFETs in parallel further reduce dissipation of the Peltier driver circuit.

3.4.3 Peltier Module Power Measurement

Measuring the power consumed by the Peltier element allows efficiency changes to be measured in real time. Ideally, 28A is the maximum current the Peltier module requires at 24V. The ±30A range of the ASC-712 current sensor (Figure 3-9) is adequate for this purpose. The ACS712 requires 5V, so a 5V linear regulator is required on the board. The current sensor output voltage must be shifted from 5V to the 0-3.3V range required by the ADC.

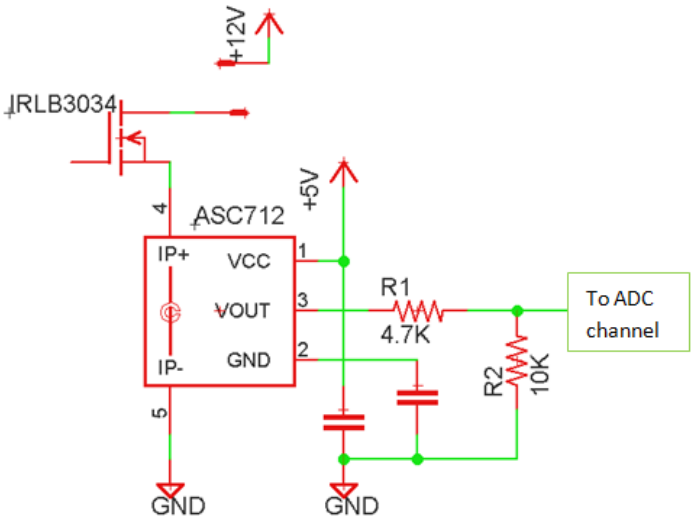


Figure 3- 9: Current sensor circuit

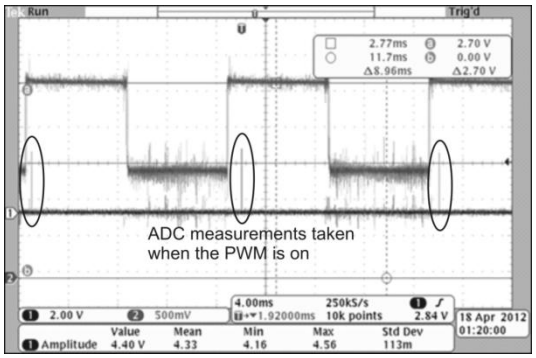


Figure 3- 10: PWM & ADC synchronization

To accurately measure the power consumed by the Peltier module, instantaneous measurement of the current and the voltage is needed. Voltage is measured at the terminals of the Peltier module to take into account any voltage drop across the supply wires. Sampling of the Peltier current and voltage takes place only when the Peltier is on.

This was achieved when the ADC sampling time was synchronized with the PWM signal shown in (Figure 3-10).

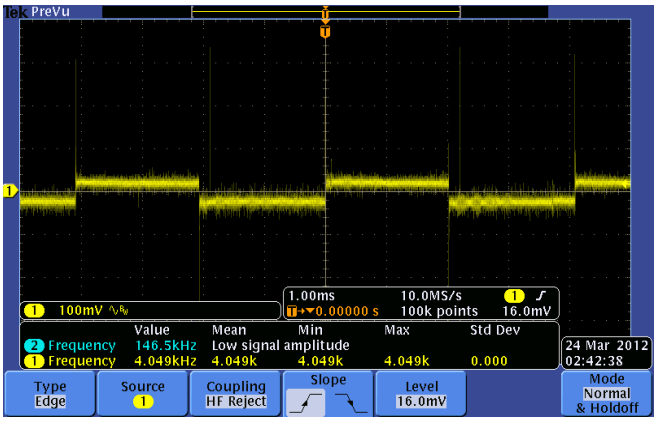
3.5 HARDWARE TESTING

Stage 1: During this stage, the designed circuits were prototyped on breadboard to make sure that each circuit functions as expected. A digital power supply was mainly used to supply the necessary voltage required by each circuit. Having different voltage levels on one circuit board is a challenge in getting the circuit to work. These voltage levels are 3.3V, 5V, 12V and 24V and they existed to meet the voltage needed by different components of the circuit. To get 24V needed by the Peltier Heater, two 12V power supplies were connected in series. These two power supplies introduced a considerable amount of noise to the system. An (LM7805) voltage regulator stepped the 12V input down to 5V for the current sensor. Similarly, an LM7833 voltage regulator provided 3.3V to the microcontroller.

The layout of the circuit was done such that the grounding of high power components was separated from the low voltage components. All Actuators were connected to the circuit board from one side using Molex Mini-fit Jr. connectors. Depending on current required by each actuator, 2, 4 and 6 pin connectors were used. Each actuator output was tested by connecting a resistive load to it. This was done to make sure that there was no short circuit existed and that the gate driver and the MOSFET functioned as expected.

Stage2: The second stage of the hardware testing was connecting actual loads to the circuit. These loads are three fans, two centrifugal pumps and a Peltier module. Most of these loads are inductive and switching such inductive loads generates destructive transient voltages unless a "freewheeling" current path is provided. Transients generate a lot of heat in the MOSFET which can damage the device. Schottky diodes were connected across both pumps to protect the switching device from being damaged by the reverse current of the inductive load.

A 25V electrolytic capacitor is connected across the power supply to filter any noise introduced by the power supply. A thick ground plane was made especially for the Peltier heater because it passes 28A through the ground plane and that introduces ground bouncing when switched by PWM signal. Figure3-11a shows a ground bouncing at the microcontroller terminal which was caused by the high power switching. This caused a lot of fluctuation in the ADC readings shown in Figure 3-11b. A 0.1uF decoupling capacitor was connected for every IC between Vcc and ground to bypass the supply inductance.



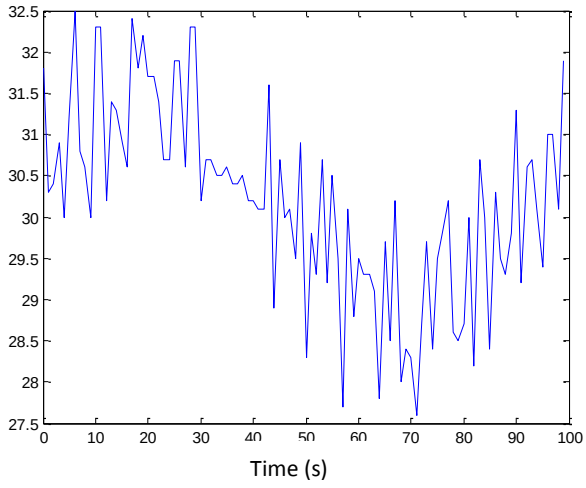


Figure 3- 11:a) Ground bouncing b) Ground bounce effect on ADC reading

3.6 SOFTWARE DESIGN & TESTING

Stage1: The software development process was comprised of four key stages. First, analog to digital Converter ADC is configured to measure the temperature of the HVAC. Six temperature sensors are connected to the ADC channels. Fixed point arithmetic was established to look up a corresponding temperature based on the ADC 10 bit reading. Timer0 was configured to generate a time interrupt of 1ms. This interrupt is useful to provide a time base line in the main program. This stage also involves tuning the temperature PI controller gain parameters to give the best response with minimum overshoot. During this stage, safety checks were implemented to protect Peltier from overheated or the pump from running dry. Figure 12 shows a flow chart of the high level design of the software.

Stage 2: In the second stage of the design, the controller is needed to be reprogrammed to observe the response of the system under various settings. This introduced a risk of destroying the controller from the physical or electrical damage (static charge that our bodies hold). The implementation of wireless communication helped us overcome this difficulty. In this stage a serial terminal program TeraTerm was utilized to transmit and receive data wirelessly using Xbee transceivers. Though it was helpful, there was no control over incoming data. Nevertheless it was the first step towards designing a Graphical user Interface.

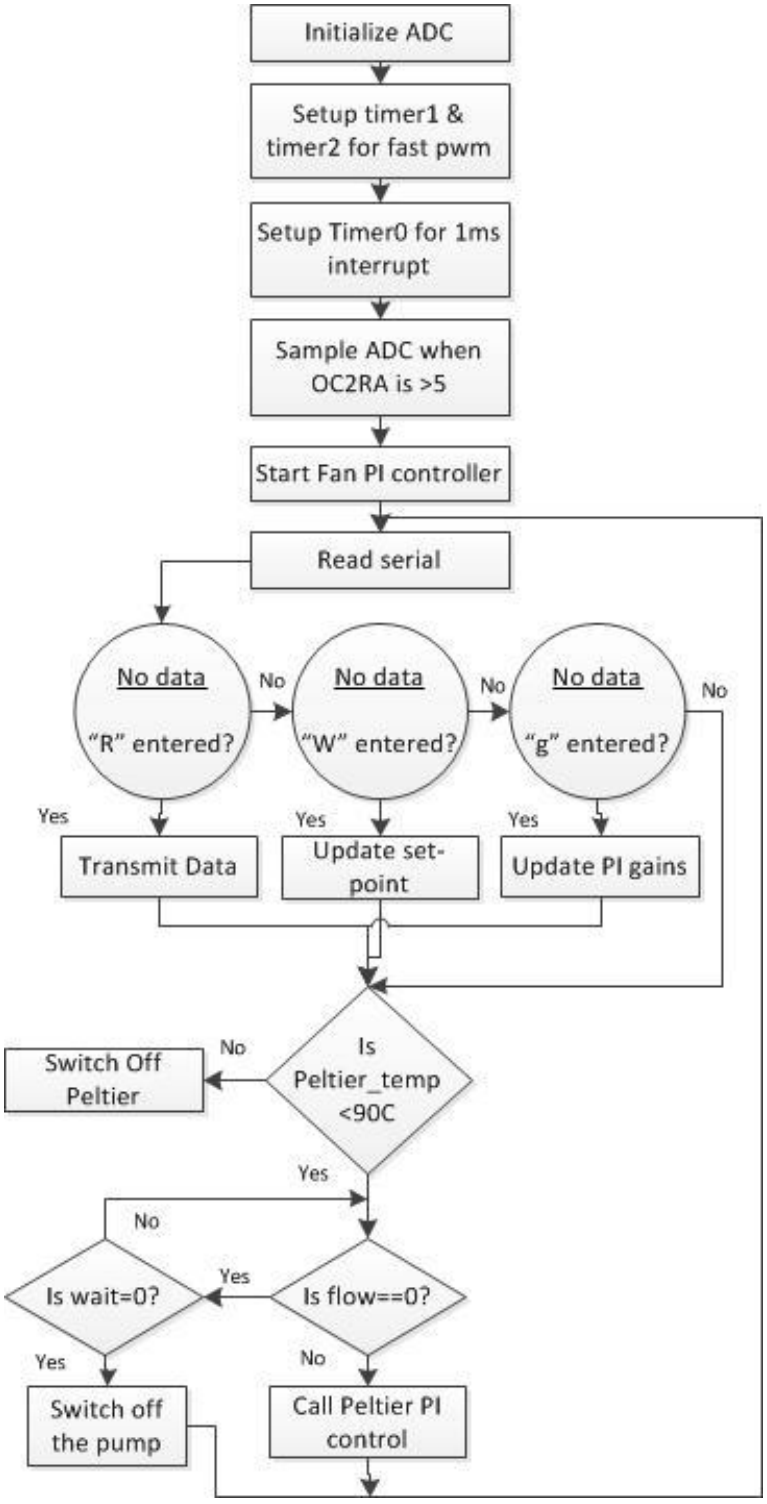


Figure 3- 12 : Software high level design

Stage 3: To facilitate communication with scaled HVAC system, GUI is designed in MATLAB. In the primary design, a control over the amount of data received was achieved. This was accomplished by restricting the MATLAB code to a certain number of user desired readings. It was not optimal compared to controlling the data flow from the controller itself, but it proved to be a good tool for debugging the performance of the system. In the later part of this design stage, temperature set-point feature was added to the system.

Stage 4: Achieving the features listed below over the data transmitted and received was the goal of this part of the design stage. All these features enhanced the control any user had over the system.

- Number of reads and interval between reads request sent to the controller
- Introducing acknowledgement signals from the receiver to confirm a successful data transfer
- Implementation of failure checks and display of appropriate error messages.

3.6.1 Final GUI Design

GUI is designed which implements the commands to acquire data wirelessly and also command the controller to reach a desired temperature in the condenser loop. The key features of the design are pointed out in figure 3-13 and explained briefly.

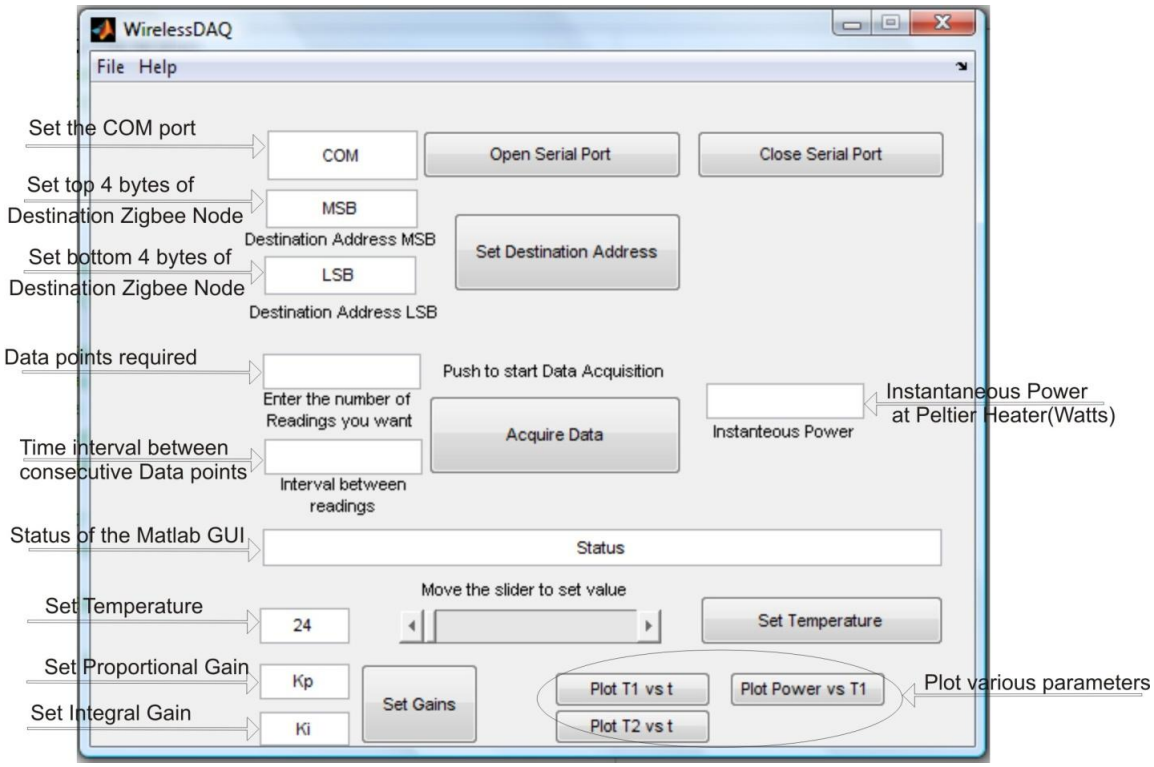


Figure 3- 13 : GUI layout design

- **Setting the COM port of the connected Zigbee Module:** As the serial port is now obsolete in most personal computers and laptops, the use of USB-Serial converters is increasing. These devices on the USB emulate a serial port and hence do not have an assigned COM port address (eg COM1, COM2 and so on). Hence on different computers we may have different port addresses. To make the GUI more user

friendly, an option was included where the user can enter the comport value and then open the serial communication using the 'Open Serial Port' button.

- **Setting the Destination Address:** The Zigbee module can hold a single destination address. In a mesh network where we have multiple nodes and want to communicate with all of them, the destination address on the Zigbee can be changed and hence communicate with the desired node.
- **Setting the number of data points required:** To implement a controlled communication, this feature of supplying the number of data points required is added. This feature commands the controller to send the data only when asked by the user and also in the quantity asked by the user.
- **Setting the time interval between reads:** Since we are dealing with controlling the temperature here, sending out data every couple of milliseconds just accumulates redundant data. Hence to control the interval between successively transmitted data point, the user is provided this feature to enter the value.
- **Setting the temperature:** This feature sets the desired inlet temperature of the heat exchanger. It is implemented using the slider bar and the text box. So, a user can update the value at any of these objects. The range of temperature that has been provided is from 24˚C to 37˚C in steps of 0.1˚C.
- **Setting the gains:** Changing the gains and seeing the response of the system is essential in choosing the best combination of gain values. This feature facilitates tuning the system without the need to reprogram the controller.

3.6.2 Decoding within the MATLAB GUI:

The following flow chart decodes the functions used in MATLAB to receive and send data.

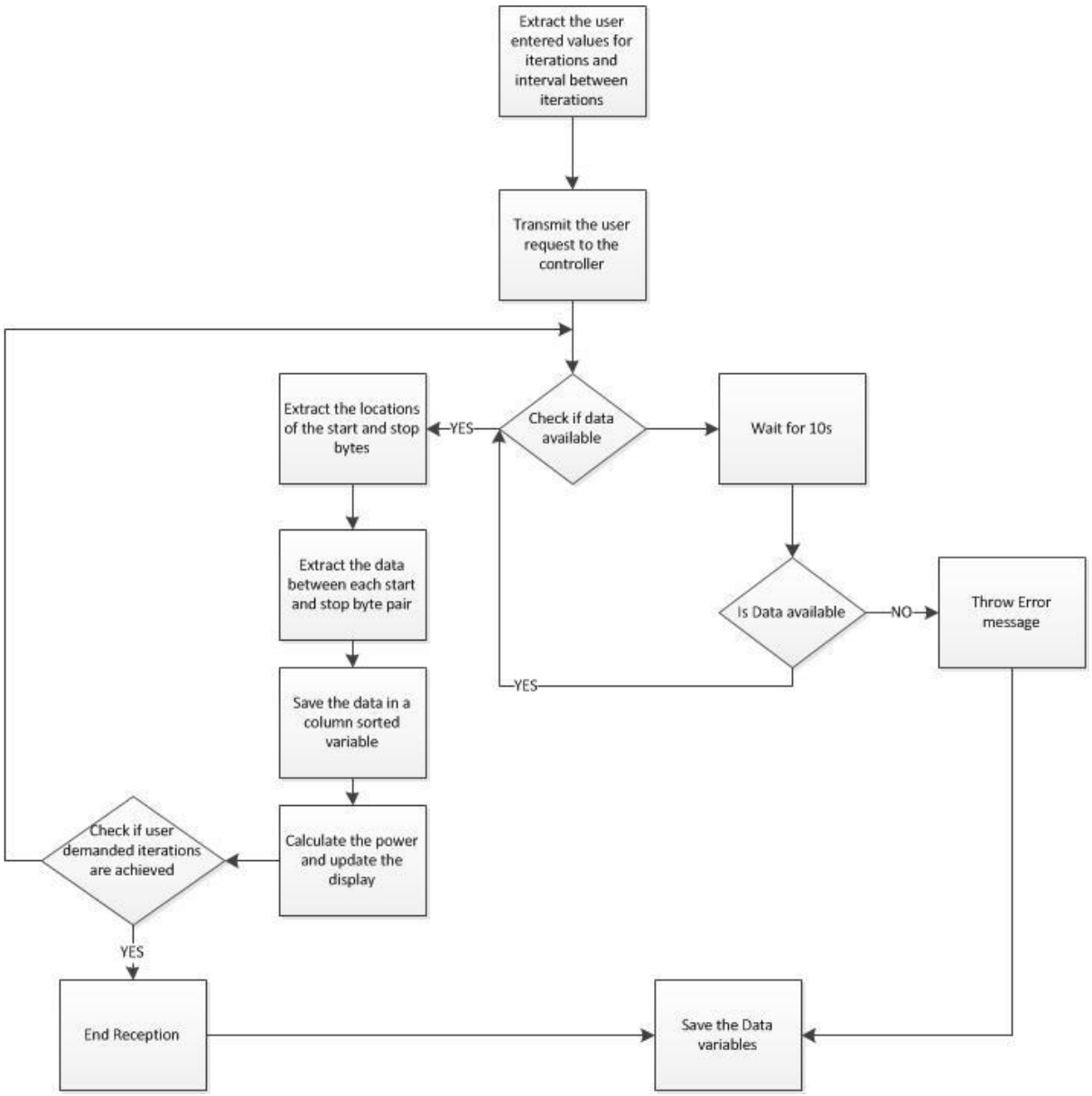


Figure 3- 14: MATLAB GUI decoding flow chart

3.6.3 Decode Logic at Controller:

In order to make the controller aware of the data received through the UART, the controller must be configured such that the UART is interrupt driven. Interrupt driven UART responds only when it receives data, then it check the type of command set to it: “R” to read data from the target, “W” to write data to the target, and “G” to tune PID parameters. Figure 3-15 summarizes the steps taken by the controller when read or write commands are received.

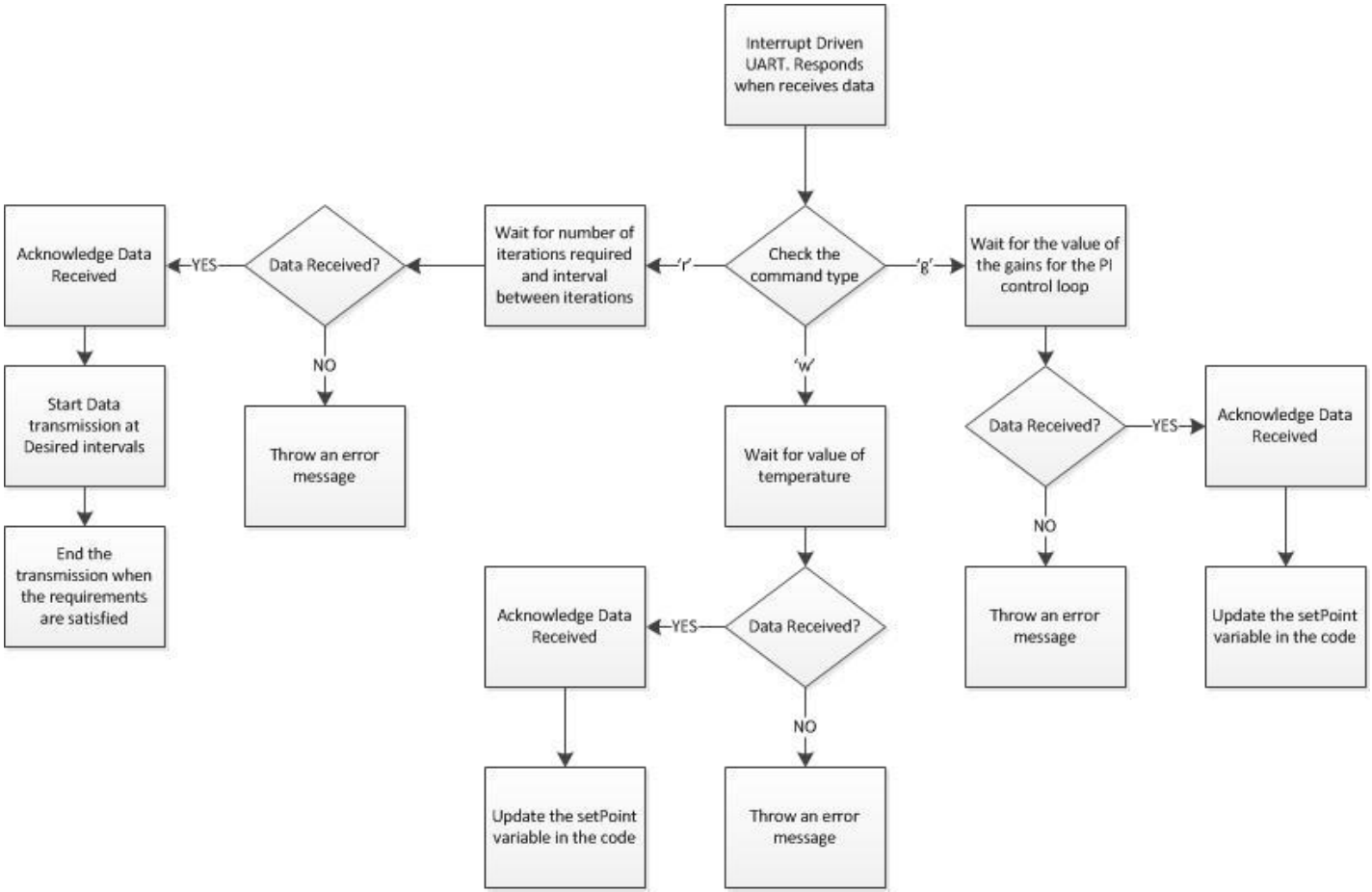


Figure 3- 15: Logic decoding in Arduino controller

CHAPTER 4: MODELING & OPTIMIZATION

4.1 INTRODUCTION

This chapter presents a centrifugal pump modeling which is an integral part of the HVAC system. Maximizing the efficiency of the centrifugal pump improves the overall efficiency of the HVAC system. Energy efficiency measures have been proven to be the most cost effective method of reducing building energy consumption, because HVAC systems are not optimized for their environment, their building load, weather forecasts, and the daily fluctuations in electricity rate [38]. New modeling methods and control systems including "model predictive control" have the potential to improve HVAC system efficiency. This chapter presents the first step towards optimizing the performance of the HVAC system by applying model-based optimization on the centrifugal pump model to maximize its efficiency. The optimum speed is used as a set-point to a local SISO PID controller to adjust the flow rate. Figure 4-1 illustrates the block diagram of the optimization routine for the centrifugal pump.

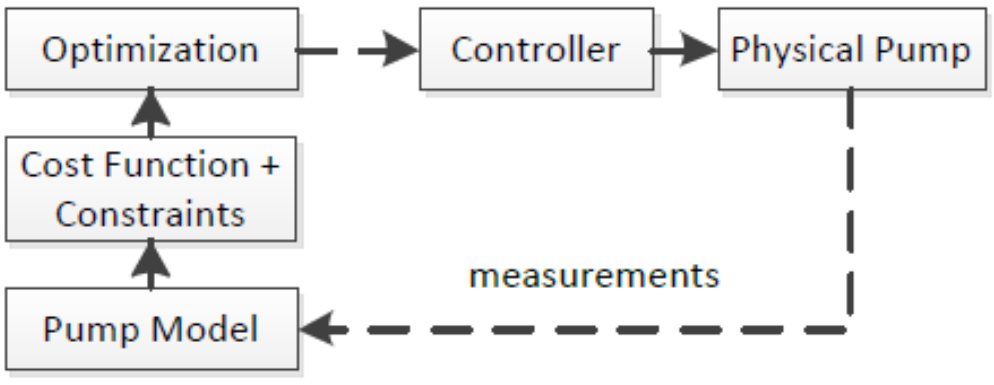


Figure 4- 1: illustrates the high level design of the complete system.

4.2 PUMP MODEL

The variable speed pump determines the flow rate through the HVAC loops. As a result, it plays a significant role in improving the efficiency of the system [20]. The pump nominal head, at a nominal pump speed N_0 is modeled as a quadratic function of the nominal volumetric flow rate:

$$\Delta p_0(q_0) = aq_0^2 + bq_0 + c$$

Where: q_0 is the nominal volumetric flow rate, and $\Delta p_0$is the nominal pressure across the pump. The efficiency curve of the pump for a nominal pump speed is also modeled as a quadratic function [20]:

$$\eta = dq_0^2 + eq_0 + f$$

Affinity laws are used to address variable speed operation of the pump. It also scales the head and the efficiency curves accordingly:

$$\frac{q}{q_0} = \frac{N}{N_0}, \frac{\Delta p}{\Delta p_0} = \frac{N}{N_0^2}^2$$

Let the ratio of the pump speed N to nominal speed N_0 denoted as L. Let N_0 be the maximum pump speed, then 0<L<1. Thus, the general pump pressure curve can be calculated using the following equation:

$$\Delta p = L^2 . \Delta p_0 (\frac{q}{L})$$

Where: $\Delta p = p_0 - p_i$. The pump power consumption is calculated using the following equation:

$$p = \frac{\Delta p . q}{\eta(\frac{q}{L})}$$

4.2.1 Model Calibration

The pump model uses polynomial curve fits in order to find the parameters of the efficiency and the head equations. Figure 4-2 illustrates the polynomial curve fits as a function of flow rate. Four flow rate points are selected to calibrate the quadratic curves. Least square fit is used to find the quadratic function parameters listed in Table 4-1.

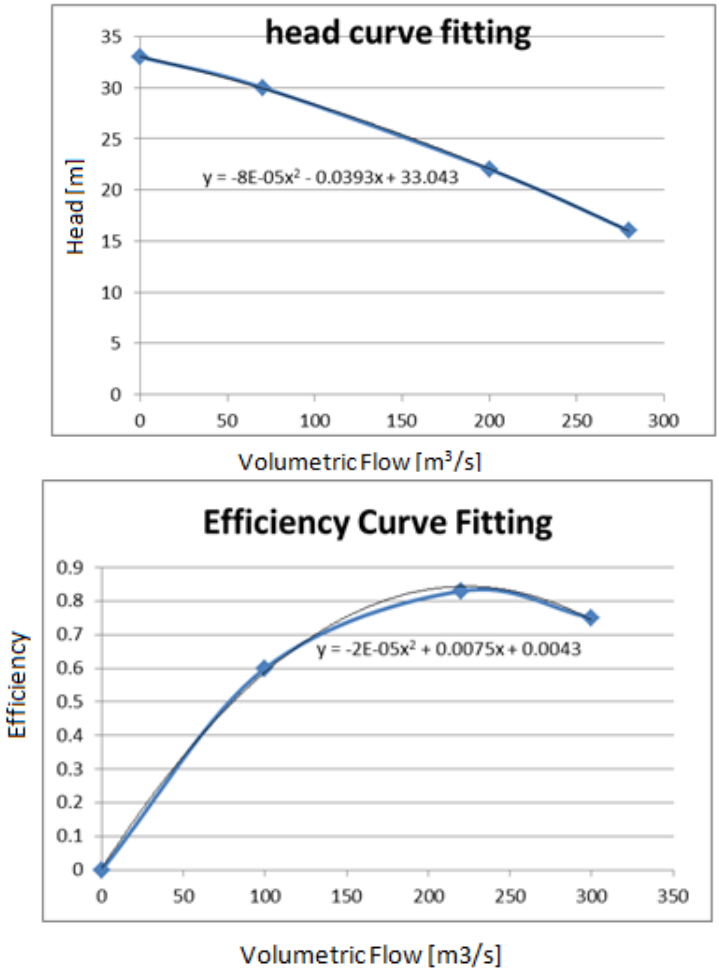


Figure 4- 2: Pump model fit based on calibration points

Name(Δp)	Value	Name (η)	Value
a	-8E^{-05}	d	-1.68E^{-05}
b	-0.0393	e	0.0075
c	33.043	f	0.0043

Table 4- 1: Calibrated parameter values parameter values for the pump

Figures 4-3 shows the pump characteristic where each of the parallel curves represents the relationship between the flow rates q, the pump generated head (Δp) for a particular motor rotation speed, N. The pump model is based on quadratic curve relating flow rate to head and efficiency and affinity laws to scale them up or down.

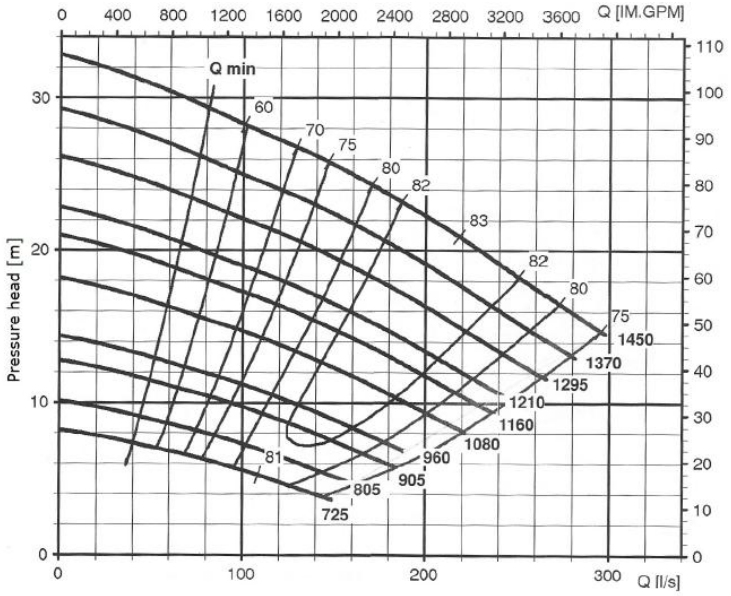


Figure 4- 3: Pump characteristic curves [7]

To confirm that our curve fit model is correct, we generate another head curve at different speed, e.g. N=1080RPM using Affinity law. Figure 4-4 shows how Affinity law scales.

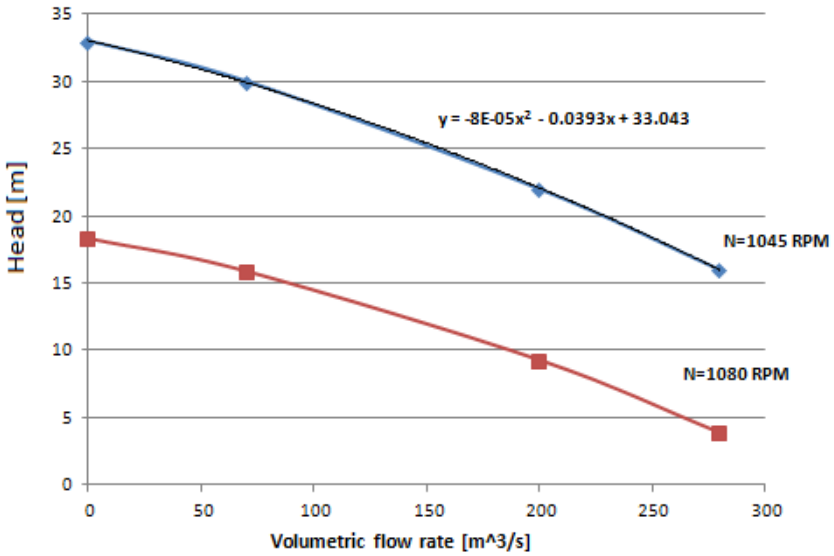


Figure 4- 4: Affinity Law scales the head curve at N=1080RPM

4.3 MODELICA

Modelica is an equation based modeling language for modeling physical systems [14]. It is mainly built from classes that are named models. Models are usually enclosed in a package and these packages are used to organize models hierarchically [6]. There is no need to solve for the variable explicitly by hand before programming [12]. Modelica decides automatically which variable to solve for if the number of equations equals the number of variables provided. A model of the centrifugal pump is developed using Modelica language shown in Figure 4-5. Dymola is the compiler used for Modelica. Modeling in Modelica and Dymola offers the advantages of having an overview over the whole model due to the object orientation and the graphical interface Dymola offers. Testing and simulation of the model are convenient using Dymola. Built-in libraries such as the building and fluid libraries were particularly useful in building our pump model.

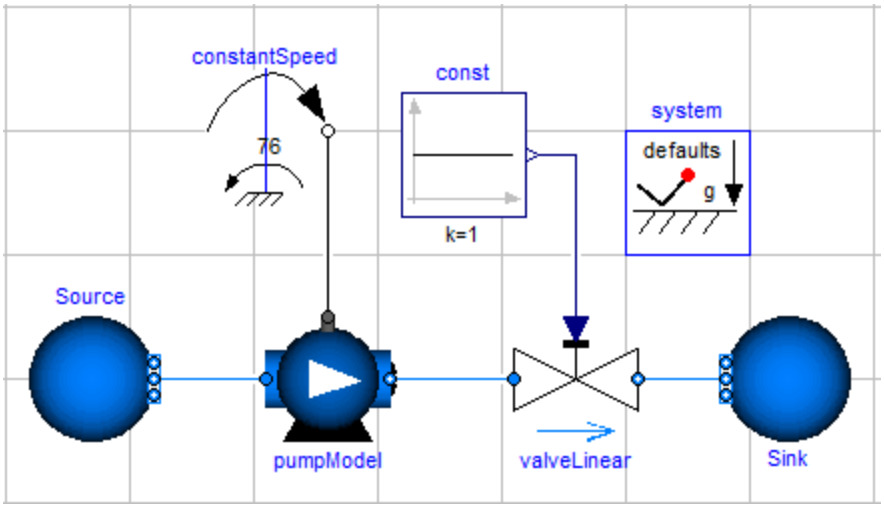


Figure 4- 5: Modelica pump model

4.4 MODEL BASED OPTIMIZATION:

Pumps consume about 20% of the world's energy consumed by electric motors [16]. The efficiency of the pump determines its performance and its energy usage. The ability to control the speed of the pump to operate at maximum efficiency allows energy savings, decreases the unscheduled downtime and increase the lifespan of the pump [16]. Optimization is mathematically the process of finding the conditions that maximizes or minimizes a function under certain constraints. The optimization of the pump is performed with the objective of maximizing its efficiency. Model based optimization maximizes pump's efficiency based on the pump's model. It requires an objective function to be maximized or minimized and a set of constraints that limits our decision variable range as shown in Figure 4-6. Model based optimization is particularly useful to avoid explicitly formulating for complex systems.

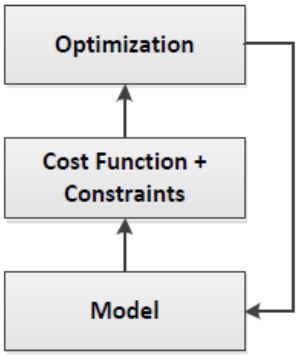


Figure 4- 6: Model based optimization

The final formulation of the optimization problem is mathematically represented by:

$$\text{Maximize } -1.68E^{-05}\, x2 + 0.0075x + 0.0043$$

$$\text{Subject to: } [725 \leq N \leq 1045]$$

$$[0 \leq x \leq 300]$$

Where: x= volumetric flow rate [L/s], N is the speed of the pump in RPM.

Fmincon function is used to minimize the negative of the efficiency as follows:

$$[x\ fval] = fmincon\ (fun, x_0, [], [], [], [], 0, 300)$$

Where: fun= ($1.68E^{-05}$ x2 - 0.0075x-0.0043).

x_0=0(L/s) initial volumetric flow rate.

Lower bounds =0, upper bounds=300 L/s, (see appendix D for the MATLAB code)

4.5 RESULTS

The maximum efficiency obtained is 84% and the corresponding flow is at 223 L/s shown in Table 4-2. This corresponds to the pump motor speed to be N=1210 RPM. Looking at the pump characteristic curve, it can be seen that the speed N=1210 rpm is the lowest speed that we still achieve the maximum efficiency the pump can offer with minimum power consumption at this particular speed. The next step in optimizing the HVAC loop performance is the integration of the centrifugal pump model with Peltier module and the heat exchanger model to formulate a complete model of the condenser loop.

RPM	rad/s	m^3/s	flow L/s	head (m)	head (pa)	efficiency
725	76	0.0812	81.2	8.2364933	80800	0.503
805	86	0.1039	103.9	10.546381	103460	0.604
905	95	0.1268	126.8	12.869520	126250	0.687
960	100	0.1405	140.5	14.259938	139890	0.728
1080	113	0.1794	179.4	18.207951	178620	0.811
1160	121	0.2057	205.7	20.876656	204800	0.838
1210	**127**	**0.2266**	**226.6**	**22.997961**	**225610**	**0.843**
1295	135	0.2561	256.1	25.985728	254920	0.825
1370	144	0.2913	291.3	29.565749	290040	0.765
1450	154	0.3332	333.2	33.813455	331710	0.639

Table 4- 2: Results obtained from Modelica pump model

CHAPTER 5: CONCLUSION & FUTURE WORK

5.1 SUMMARY OF CONTRIBUTIONS

The main contribution of this thesis is a functioning scaled HVAC system that is controlled wirelessly via Zigbee protocol. A wireless sensor actuator network is designed and implemented to both send and receive data from the scaled HVAC. Wireless tunable PI controllers are successfully implemented and tested on Mega 328p microcontroller. Power consumption of the Peltier is measured, calibrated and indicated on the GUI interface. This will work as guidance on how to accurately measure other actuators power consumption. The circuit design is prototyped and tested for future PCB manufacturing. A summary of the results obtained from each part of the system is listed as follows:

- Controller software algorithm is successfully implemented on Arduino microcontroller including ADC configuration, 1ms interrupt generation, fixed point arithmetic for temperature measurement, PWM signal generation, multiple PI loops, wireless serial communication and safety checks using finite state machine.
- Electronic circuits are designed and built using appropriate protective safety measures and thick, short ground plane to protect against ground bouncing issues. Figure 5-1b indicates the effects of ground bouncing on the sensor measurements. Figure 5-2a illustrates a less effect of ground bouncing when thick, short ground plane is implemented.
- Temperature Sensors are calibrated by immersing the Thermistor sensor in to a cup containing boiling water. As water cools down, temperature measurements were

taken using an accurate thermocouple sensor and the corresponding resistance of the Thermistor is recorded. I used one look up table to get the temperature readings for all Thermistor sensors; instead one could have included a look up table for each sensor. However, execution time and memory space needs to be considered.

- The ASC 712 current sensor is calibrated using an accurate current sensor provided in the lab. Figure 5-1a plots the actual Peltier current and the current sensor output voltage and used the quadratic fit equation to convert from voltage to current.
- A two way communication between HVAC system and the PC is implemented via Zigbee protocol. A GUI is developed in MATLAB to receive and transmit data on demand and to facilitate tuning PID and changing the set point without the need to reprogram the microcontroller.
- A satisfactory performance of the PI control loops is achieved by a well-tuned PI loops and improvements of sensor ground bouncing issues (Figure 5-2a). SISO demonstrated a very fast stable response and is considerably easy to tune. Multiple SISO PI loops gave a reasonable performance to control highly coupled thermal states in the system.
- Chapter4 covered the modeling process of a centrifugal pump. An equation based modeling of the pump is obtained and a Modelica pump model is created on Dymola platform. This model is called from Matlab to maximize the pump's efficiency to find the optimal operation speed at which the pump performs best.

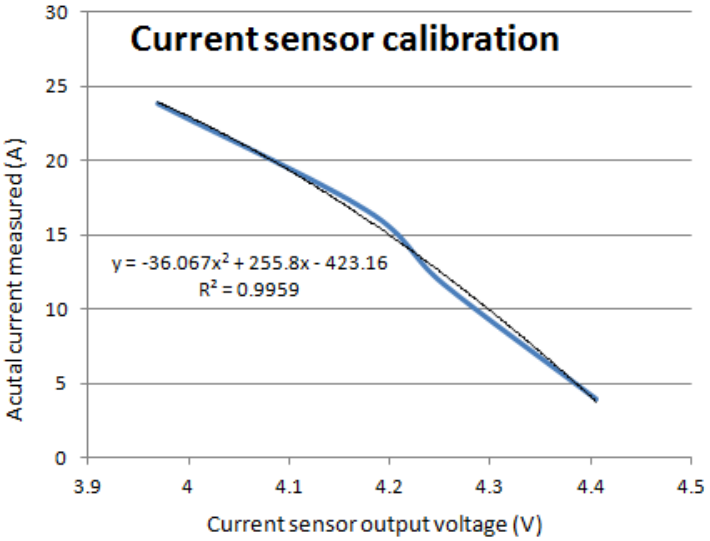


Figure 5- 3a: Current sensor calibration

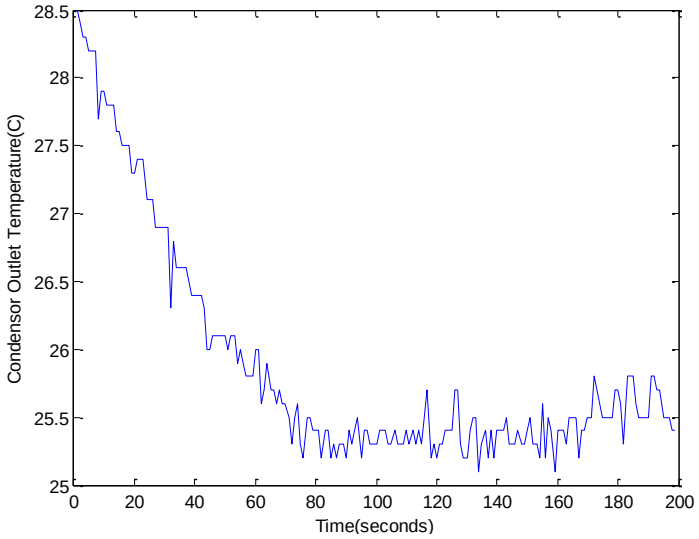


Figure 5-1b): Ground bouncing effects on the signal

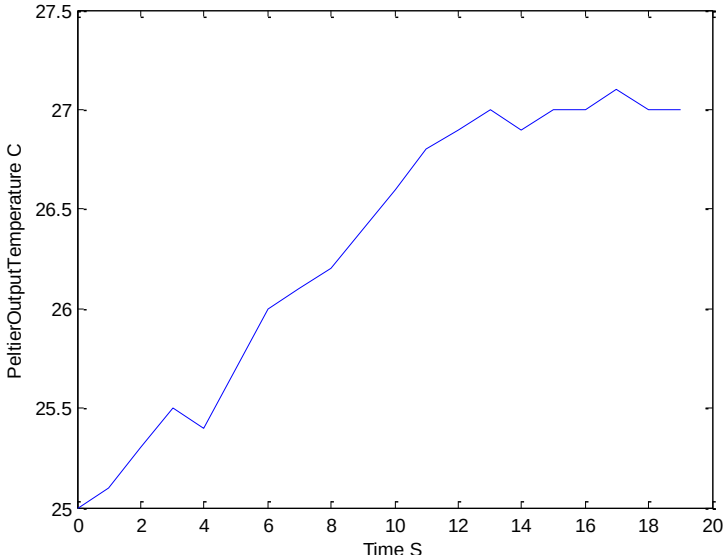


Figure 5- 1: a) Peltier transient response in SISO loop

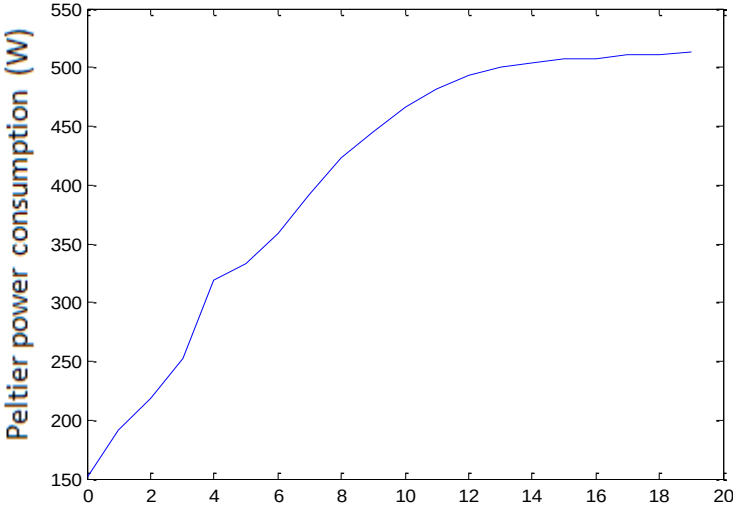


Figure 5- 2: b) Peltier power consumption

5.2 CONCLUSION

In the past, buildings controller design was mainly concerned with functionality and performance. Reduction of energy consumption is a relatively recent concern that has been investigated using computer simulation only. Building a complete test bed allows testing of a building design prior to building. It also facilitates repeatable laboratory experiments to validate computer simulation assumptions.

In this work, I designed and built the hardware and the software algorithm needed to operate and control the scaled HVAC system which will be used to condition a scaled building envelope in the test bed. The modular design of the signal conditioning circuits, actuator driver circuit, and power regulation and measurement allow manufacturing of PCB circuits. This facilitates the design of a larger HVAC system which will be used to condition the weather simulation enclosure in the Lab. In the software part, PID algorithm has been successfully implemented to control individual thermal states. A generic PID code is created such that any other thermal states can be controlled as well. Controlling these individual states enables us to find the best combination of states at which best performance is obtained and with least delay caused by the nested PID loops.

When dealing with switched mode operation on devices, it is necessary to have a sufficiently large ground plane to suppress the transients. A thick, short ground plane is needed especially in high power circuits. If the ground plane is not sufficiently thick, frequent switching causes the ground level to bounce and hence affect the performance of the microcontroller and ADC readings. Introducing ground bouncing noise in the feedback sensor introduces instability in the feedback control and may lead to unexpected controller behavior.

The choice of the controller P, PI, PD or PID depends on the type of the plant. Temperature controller is usually classified as type-0 system; i.e has no integral term built-in in the forward path. This type of a system requires the implementation of PI controller which will provide steady state accuracy. Note that a type-0 system under discrete time, increasing K_p can indeed oscillate due to the delay between samples.

5.3 SAFETY

This project involves a high DC power application. Hence it is very important to keep the safety concerns in mind while interacting with the system.

Peltier module is sensitive to the current. Hence the duty cycle of the PWM applied defines the average power being relayed to the device. Whenever the control for the Peltier module is revised in the controller code, it is a safe practice to check the output of the controller pin controlling the device on the oscilloscope by connecting a simple resistive load. This gives an idea about the performance of the system and helps us determine how the actual device will react to such an input. Even a minor programming error can lead to overheating the Peltier and destroy the peripherals as well.

Not only malfunctions related to the Peltier can cause it to overheat. The performance and control over the peripherals can also cause the system to go haywire. For instance, if the condenser pump is not working, the Peltier will not be able to transfer the heat to the heat exchanger and hence heat up very quickly. Same is the case with the failure in the control of the heat exchanger fans. So as a standard procedure it is good to check the control signals on the oscilloscope before hooking up the devices.

Solution implemented:

- Check for the temperature at the Peltier heater. If it exceeds the limit of 90˚C, turn off the Peltier. It cannot restart until reset manually. This enables the user of the system to know that there is some problem in the system.
- The second check is whether the pump in the loop is working or not. If it is working without any fluids, it may burn out the coil of the pump. Hence we check the flow in the loop and if it is zero, turn off the pump and the Peltier module. As the previous check, this is a manual reset operation to alert the user about any issues with the system.

Current Sensor IC ACS 712: This is a surface mount component designed to carry 30 A current. But the connectors on which it rests may not be as capable as the IC. So there is a need to cool down the IC to avoid burn out.

Solution: Currently I have the cooling fans within the power source we are using, directed towards the circuit board. This basically cools down everything there in and works as a temporary solution. It is recommended to build an enclosure box for the electronics and install a small fan on the enclosure.

Multi-value Voltage supplies for various components: On this board, we have components working at 3.3 V, 5 V, 12 V and 24 V. Hence it is very important to take care not to short out two different voltage values while trying to use oscilloscope, multi-meter or something as simple as hook up wires.

Solution we have:

- We used protective power and signal connectors, those which can be plugged in one direction only. Hence there are no chances of inverting the Vcc and ground terminals.
- Avoiding the shorting of different voltage sources is up to the skills and care taken by the user.

5.4 DESIGN CONSIDERATION AND POSSIBLE IMPROVEMENTS

Optical Isolation:

As discussed before, grounding can be an issue on the performance of the low power devices (controller, sensors etc.) when using along with high power switching devices (pumps, heaters etc.). Hence it would be a good idea to isolate the low power and high power devices to stabilize the performance of both. An opto-isolator (capable of accommodating the desired bandwidth of the switching signal) would serve this purpose by transferring the low power switching signal optically at the high power end. This would practically isolate the two power levels. IC's are readily available for this purpose. Current design does not have this feature but it would be good to have it during future improvements.

Appendix A: Parts sources and cost

No.	Part	Unit Cost	Quantity	URL
1	Arduino Boards	$25.00	3	http://www.sparkfun.com/products/10116
2	Xbee Wireless Modems	$22.00	3	http://www.sparkfun.com/products/8665
3	INS- FM18 Flow meter sensor	$30	2	http://www.koolance.com/water-cooling/product_info.php?product_id=1170
4	Peltier Heater/Cooler	$56.50	2	http://www.shop.customthermoelectric.com/searchquick-submit.sc;jsessionid=49BADFC26E6C47EDBF65E2A8D246654D.qscstrfrnt01?keywords=400
5	Swiftech MCP655 circulating Pump	$76.00	1	http://www.sidewindercomputers.com/swmc12vdcpu.html
6	Swiftech MCP-35X pump	$110.00	1	http://www.sidewindercomputers.com/swmc12vdcpu1.html
7	LCD display screen	$13	1	http://www.sparkfun.com/products/255
8	Power Supply for Peltier Cooler 24 V	$210.00	2	http://www.trcelectronics.com/Meanwell/se-1000-12.shtml
9	$2	$2.00	4	http://www.datasheetcatalog.com/datasheets_pdf/M/C/P/6/MCP6004.shtml
10	lm2937et Current regulator	$1.00	1	http://www.datasheetcatalog.com/datasheets_pdf/L/M/2/9/LM2937ET-12.shtml
11	Water Blocks	$79	2	http://www.customthermoelectric.com/Water_blocks.html
12	IC Gate Driver non-inverting 8Dip	$2	2	IXDN604PI
13	MOSFET	$4	5	IRLB3034PBF
14	3.3 voltage regulator	$1		www.digikey.com
15	7805 voltage regulator	$1		www.digikey.com
16	12V votlage regulator.	$1		www.digikey.com
17	ASC current sensor	$4		www.digikey.com
18	Condenser radiator	$70	1	http://www.koolance.com/water-cooling/product_info.php?product_id=813
19	Chiller loop exchanger	$39	1	http://www.frozencpu.com/products/5323/ex-rad-106/Black_Ice_GT_Stealth_120_Radiator_-_Blue.html?tl=g30c95

Appendix B: Microcontroller code

```
/*……written by Fisal Sayed……*/

#include <avr/io.h>

#include <stdlib.h>

#include <avr/interrupt.h>

#include "lookup.h"

/*--------Init global Variable------*/

volatile unsigned char timer_value;

volatile int pulses_per_second,pulses_per_second_2, wait=5000;

int e,error, setpoint = 300,setpoint2=290, inlet_temp, air_temp,cond_out,
peltier_temp,temperature_filtered;

unsigned char flag_hotloopPump=0,flag_peltierHeat=0;

volatile int adc_value[8];

static unsigned char old_pin_state=0, old_pin_state_2=0;

int adc_value0, output;

int adc_value1;            //Variable used to store the value read from the
ADC

uint16_t adc_value2;

char buffer[5];               //Output of the itoa function

uint8_t i=0;                //Variable for the for() loop11

void adc_init(void);            //Function to initialize/configure the ADC

uint16_t read_adc(uint8_t channel);    //Function to read an arbitrary
analogic channel/pin

float LPM ;

const int tempAdc[] = {   129, 167, 215, 277, 314, 354, 397, 444, 493, 543,
595, 646, 696, 744, 788, 828, 864, 895 };

const int tempCx10[] = { 1000, 900, 800, 700, 650, 600, 550, 500, 450, 400,
350, 300, 250, 200, 150, 100,  50,   0 };




Table1d temperatureTable =

{

  18,       /* Number of columns */
```

```
  tempAdc,  /* Input data */
  tempCx10, /* Output data */
};
/*---------------Peltier PI controller-----------------------*/
struct PIControl
{
  int kp;              /**< Proportional gain constant */
  int ki;              /**< Integral gain constant */
  unsigned char shift; /**< Right shift to divide */
  int max;             /**< Maximum value */
  int min;             /**< Minimum value */
  long i;              /**< Current integrator value */
};
int pi_control (struct PIControl *p, int e)
{
  bool int_ok;      /* Whether or not the integrator should update */
  long new_i;       /* Proposed new integrator value */
  long u;           /* Control output */
    /* Compute new integrator and the final control output. */
  new_i = p->i + e;
  u = (p->kp * (long)e + p->ki * new_i)>> p->shift;
int_ok = true;
       /* Positive saturation? */
  if (u > p->max)
  {u = p->max;  /* Clamp the output */
  if (e > 0) /* Error is the same sign? Inhibit integration. */
    {int_ok = false;
    }  }
  /* Repeat for negative sign */
  else if (u < p->min)
  {
```

```
    u = p->min;
    if (e < 0)
    {
      int_ok = false;
    }
  }
/* Update the integrator if allowed. */
  if (int_ok)
  {
    p->i = new_i;
  }
  return (int)u;
}
void pi_control_init (struct PIControl *p)
{
  p->i = 0L;
}
struct PIControl Peltier_pi =
{
  20,  // kp gain
  1,   //ki gain
  1,   // right s hift to divide
  255,    // max
  10,    // min
};
void control_Peltier (void)
{
  e =(setpoint - inlet_temp);
    output = pi_control(&Peltier_pi, e);
  OCR2A = output;                   // output of PID controller to D11 on
Arduino
}
```

```
/*-------------------Fan PI control-----------------------------------*/
struct PI_Control
{
  int kp;              /**< Proportional gain constant */
  int ki;              /**< Integral gain constant */
  unsigned char shift; /**< Right shift to divide */
  int max;             /**< Maximum value */
  int min;             /**< Minimum value */
  long i;              /**< Current integrator value */
};
//pi control function
int pi_control (struct PI_Control *p, int e)
{
  bool int_ok;      /* Whether or not the integrator should update */
  long new_i;       /* Proposed new integrator value */
  long u;           /* Control output */
    /* Compute new integrator and the final control output. */
  new_i = p->i + e;
  u = (p->kp * (long)e + p->ki * new_i)>> p->shift;
int_ok = true;
       /* Positive saturation? */
  if (u > p->max)
  {u = p->max;  /* Clamp the output */
  if (error > 0) /* Error is the same sign? Inhibit integration. */
    {int_ok = false;
    }  }
  /* Repeat for negative sign */
  else if (u < p->min)
  {
    u = p->min;
    if (error < 0)
```

```
    {
      int_ok = false;
    }
  }
/* Update the integrator if allowed. */
  if (int_ok)
  {
    p->i = new_i;
  }
  return (int)u;
}
void pi_control_init (struct PI_Control *p)
{
  p->i = 0L;
}
//Control Varaibles for the PI structure
struct PIControl fan_pi =
{
  20,  // kp gain
  1,   //ki gain
  1,   // right s hift to divide
  255,    // max
  0,    // min
};
//PI controller for the heat exchanger fans on D9
void control_fan (void)
{
  error =-(setpoint2 - cond_out);
  output = pi_control(&fan_pi, error);
  OCR1A = output;        // output of PID controller to D9 on Arduino
} /*-------------------------------------------*/
```

```
#define TEMP_FILTER_SHIFT 2
#define TEMP_FILTER_SIZE 4
int temp_filter[TEMP_FILTER_SIZE];
int temp_filter_sum;
unsigned char temp_filter_idx;
volatile int time1,pause=0,iterations;
volatile unsigned char flag=0, flag_transmit=0;
int main(void)
{
  unsigned char timer_saved, timer_copy;
  int data, value=0, identifier;
    Serial.begin (9600);
  adc_init();        //Setup the ADC
  DDRD |= 0x13;  //0b10010  D2 & D4 on
  DDRB=0x30;  //0b110000
  // PWM set up using timer2
  DDRB |= 0x0E;  // PD5 is now an output pin  D9 on arduino
  DDRD |= 0x88;
   OCR2A =0;     // D11 Connected to the peltier Heater
  OCR2B=255;      //D3 Connected to the pump on the hot loop
  TCCR2A |= (1 << COM2A1) | (1<<COM2B1);   // set non-inverting mode
  TCCR2A |= (1 << WGM21) | (1 << WGM20);   // set fast PWM Mode
  TCCR2B |= (1 << CS20)| (1 << CS22);
  OCR1A = 128;     // set D9 Connected to the heat exchanger fans
  OCR1B=0;  //D10
  TCCR1A |= (1 << COM1A1) | (1<<COM1B1);   // set non-inverting mode
  TCCR1A |= (1 << WGM12) | (0 << WGM11) | (1 << WGM10);   // set 8 bit fast
PWM Mode
  TCCR1B |= (1 << CS12);
  // set 1ms interrupt using timer0
  TCCR0A = 0XC2;                            //11000010  // Set the Timer Mode
to CTC
```

```
  OCR0A = 124;                                  // Set the value that you want to
count to

  TIMSK0 =0X02;                                 // Enable match interrupts

  CCR0B = 0x03;                                // set prescaler to 64
8Mhz/64=7.815 khz.

  sei();

  for(;;)

  {         //Our infinite loop

    timer_copy = timer_value;

    PORTD |= 0x10;

    //check for commands from Matlab GUI

     if(Serial.available()>0)

     {

      identifier=Serial.read();

      switch(identifier)

       {

         case 'r':

         data_transmit();

         identifier=0;

         break;

          case 'w':

         update_setpoint();

         identifier=0;

         break;


         case 'g':

         update_gains();

         identifier=0;

         break;

       }

    }

    if (((unsigned)(timer_value - timer_saved) & 0x7F) >= 100)
```

```
    {
      timer_saved=timer_copy;
      //look up the temperature values from the table
      lookup1d(&temperatureTable, adc_value[3], &inlet_temp);
      lookup1d(&temperatureTable, adc_value[4], &cond_out);
      lookup1d(&temperatureTable, adc_value[5], &air_temp);
      lookup1d(&temperatureTable, adc_value[1], &peltier_temp);
      control_fan();
      //safety checks for peltier for overheat or hot loop pump not working
      if(peltier_temp<=900 && flag_hotloopPump==0 && flag_peltierHeat==0)
        control_Peltier();
      else
        {
        //turns off the peltier in case of overheat or pump failure
        //demands a manual reset for safety purpose
        OCR2A=0;
        flag_peltierHeat=1;
        }
    }
    //check if the flow is zero in the hot loop pump for atleast 5 seconds
    if(pulses_per_second==0 && wait==0)
    {
        //turn off the pump if no flow is detected for 5 seconds continously
        //demands a manual reset for safety concerns
        PORTD=(0<<PIN2);
        flag_hotloopPump=1;
    }
    else
      wait=5000;

    //sync the ADC reads with PWM on time to get instantaneous values of
current and voltage
```

```
if(TCNT1L>(OCR2A-5))
{
  flag=0;
  PORTD=(0<<PIN7);
}
//transmits when demanded at user defined intervals
if(flag_transmit==1 && iterations>0)
  {
  if(time1==0)
    {
    Serial.print("~");
    Serial.print(inlet_temp);
    Serial.print ("~");
    Serial.print ("~");
    Serial.print (cond_out);
    Serial.print ("~");
    Serial.print ("~");
    Serial.print(pulses_per_second,DEC);
    Serial.print ("~");
    Serial.print ("~");
    Serial.print(pulses_per_second_2,DEC);
    Serial.print ("~");
    Serial.print ("~");
    Serial.print(adc_value[6],DEC);
    Serial.print ("~");
    Serial.print ("~");
    Serial.print(adc_value[7],DEC);
    Serial.println ("~");
    if(iterations==1)
      flag_transmit=0;
    --iterations;
```

```
        time1=pause;
        }
      }
     //LPM=0.307*pulses_per_second;
     }    //end for
}    //end main


ISR (TIMER0_COMPA_vect)  // timer0 Compare match interrupt
{
  static int prescaler;
  static int pulse_count, pulse_count_2;
  unsigned char new_pin_state = PINB & 0x10;  // Read PORT B pin 4 - flow
sensor on hot loop
  unsigned char new_pin_state_2 = PINB & 0x20;// Read PORT B pin 5 - flow
sensor on cold loop
  static unsigned char adcChannel = 0;
    //event to be executed every 1ms here
  ++timer_value;
    if (old_pin_state != new_pin_state)
  {
    if ((old_pin_state == 0) && (new_pin_state != 0))
    {
      ++pulse_count;
    }
    old_pin_state = new_pin_state;
  }
   if (old_pin_state_2 != new_pin_state_2)
  {
    if ((old_pin_state_2 == 0) && (new_pin_state_2 != 0))
    {
      ++pulse_count_2;
    }
```

```
    old_pin_state_2 = new_pin_state_2;
  }
    ++prescaler;
  if(time1>0) --time1;
  if(pulses_per_second==0) --wait;
  if (prescaler == 1000) //update the flow value each second
  {
    pulses_per_second = pulse_count;
    pulses_per_second_2=pulse_count_2;
    pulse_count=0;
    pulse_count_2=0;
    prescaler = 0;
  }

//read ADC when PWM is on to get instantaneous
if(TCNT1L>3 && TCNT1L<(OCR2A-5))
    {
      if (0 == (ADCSRA & (1 << ADSC)) && flag==0)
        {
        PORTD=(1<<PIN7);
        flag=1;
        adc_value[adcChannel] = ADCW;
        /* Go to the next channel.  Wrap past 8. */
        adcChannel = (adcChannel + 1) & 0x07;
        ADMUX = (ADMUX & 0xF0) | adcChannel;
        /* Start the next conversion */
        ADCSRA |= (1<<ADSC);
        PORTD=(0<<PIN7);
        }
    }
  if(TCNT1L>(OCR2A-5))
```

```
    {
       flag=0;
      PORTD=(0<<PIN7);
    }
}
//end the timer 0 compare match interrupt


//init the ADC
void adc_init(void)
{
  PRR &= ~0x01;                          /* Disable ADC power-down logic */
  ADCSRA |= ((1<<ADPS2)|(1<<ADPS1));    //8Mhz/64 = 125Khz the ADC reference
clock
  ADMUX = 0x40; /* Use 5V reference, start at channel 0 */
  ADCSRA |= (1<<ADEN);                 //Turn on ADC
  ADCSRA |= (1<<ADSC);                 //Do an initial conversion because this
one is the slowest and to ensure that everything is up and running
}//end ADC init
//Data trasmit sets the variable for transmit operation
void data_transmit(void)
{
      iterations=getdata(); //gets the iteration value
      pause=getdata(); //gets the iteration interval
      time1=pause;
      flag_transmit=1; //sets the flag for transmit
}
//end data_transmit
void update_setpoint(void)
{
  setpoint=getdata();  //gets the new setpoint
  Serial.println("OK");//acknowledges successful update
}//end updatae_setpoint
```

```
void update_gains(void)
  {
  //get the new gain values
  Peltier_pi.kp=getdata(); //fan_pi is used to control kp for fan controller
  Peltier_pi.ki=getdata();//fan_pi is used to control ki for fan controller
    Serial.println("OK"); //acknowledges successful update
  } //end update_gains
int getdata(void) //gets data from UART-Terminating Character is Linefeed '\n
{
  int value;
  value=0;
  while(1)
    {
     int data=Serial.read();
     if(data=='\n')
        {
        return value;
        break;
        }
     else if (data>47 && data<58)
        {
         data=data-0x30;
         value=value*10+data;
        }
    }
   }  //end getdata()
```

Appendix C: GUI MATLAB code

```
% Written by: Fisal Sayed & Nirav Patel.  April /2012
function varargout = WirelessDAQ(varargin)
% WIRELESSDAQ M-file for WirelessDAQ.fig
%      WIRELESSDAQ, by itself, creates a new WIRELESSDAQ or raises the
existing
%      singleton*.
%
```

```
%      H = WIRELESSDAQ returns the handle to a new WIRELESSDAQ or the handle
to
%      the existing singleton*.
%
%      WIRELESSDAQ('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in WIRELESSDAQ.M with the given input
arguments.
%
%      WIRELESSDAQ('Property','Value',...) creates a new WIRELESSDAQ or
raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before WirelessDAQ_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to WirelessDAQ_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help WirelessDAQ

% Last Modified by GUIDE v2.5 26-Apr-2012 14:44:06

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @WirelessDAQ_OpeningFcn, ...
                   'gui_OutputFcn',  @WirelessDAQ_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before WirelessDAQ is made visible.
function WirelessDAQ_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to WirelessDAQ (see VARARGIN)

% Choose default command line output for WirelessDAQ
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
```

```
set(handles.slider1,'Value',24);   %set default values for slider
set(handles.sliderVal,'String','24');%set default values for Display for
slider
setappdata(handles.sliderVal,'sliderVal','24'); %save the value of
temperature
% UIWAIT makes WirelessDAQ wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = WirelessDAQ_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%-------------------------Data Acquisition Part--------------------------%

%---User Sets the Iteration value in the edit box---%
function iterationVal_Callback(hObject, eventdata, handles)
% hObject    handle to iterationVal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of iterationVal as text
%        str2double(get(hObject,'String')) returns contents of iterationVal
as a double

iterations=get(hObject,'string');   %get the user entered value
index=strfind(iterations,'.');
if(isempty(index) && str2double(iterations)> 0) %check for non negetive
integer input
    %save if qualifies the requirements
    setappdata(handles.iterationVal,'itrVal', get(hObject,'String'));
    %save ('data.mat','iterations'); %save the demanded data values in a file
else
    %throw an error message if not compatible
    msgbox('Incompatible Input. Please enter an Integer Value greater than
0.','Input Error','Warn');
end

% --- Executes during object creation, after setting all properties.
function iterationVal_CreateFcn(hObject, eventdata, handles)
% hObject    handle to iterationVal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
%---User sets the interval between consecutive reads ---%

function readInterval_Callback(hObject, eventdata, handles)
% hObject    handle to readInterval (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of readInterval as text
%        str2double(get(hObject,'String')) returns contents of readInterval
as a double
val=get(hObject,'String');
%fprintf(val);
if(isempty(strfind(val,'.')) && str2double(val)>=200) %check for non negetive
integer input
    %save if qualifies the requirements
    setappdata(handles.readInterval, 'interVal',val);
else
    %throw an error message if not compatible
    msgbox('Incompatible Input. Please enter integer values from 200 to
32676.','Input Error','Warn');
end



% --- Executes during object creation, after setting all properties.
function readInterval_CreateFcn(hObject, eventdata, handles)
% hObject    handle to readInterval (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ------Starts Data Acquisition--------%

function AcquireData_Callback(hObject, eventdata, handles)
% hObject    handle to AcquireData (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%resets all the communication devices using the serial port
instrreset;
clc;    %clear command window
%clear all;  %clear old versions of variables
try             %try-catch error detection mechanism starts
fid=load('serial.mat','s'); %load the serial object created in SerialOpen
%load the number of iterations entered by the user
no_iterations = getappdata(handles.iterationVal,'itrVal');
%itrVal=str2double(no_iterations);
%s=serial('COM1');   % the communication port
%save('Data.mat','s');
```

```
%set(s,'terminator','CR'); %configure the terminating character in the
communication
fopen(fid.s);   %open the serial port

%-----init variables-----%

%data=[];    %collects all the incoming serial data
iteration=1; %init iteration
raw_data=zeros(1000,8);  %init the data matrix
Power=zeros(str2double(no_iterations),1);
a=0;    %total number of bytes received
Vref = 3.3; %Voltage reference for ADC
VScale=9.2; %Voltage divider scaling factor for voltage sensor
IScale=14.7/10;%Voltage divider scaling factor for current sensor

%get the interval value from the interval object
interval=getappdata(handles.readInterval,'interVal');
%Send the Data transmission command to the controller
fprintf(fid.s,'r');
set(fid.s,'terminator','LF');
%send the number of iterations
fprintf(fid.s,no_iterations);
pause(0.2);
%send the interval between iterations
fprintf(fid.s,interval);
%Update the status
set(handles.Status,'String','Receiving...');
pause(2);

while(1)    % check for incoming data
  pause(str2double(interval)/1000);
  b=fid.s.bytesavailable();     %find the bytes available
  j=1;
  if(b>0)   %if available
    a=a+b;  %increment the global count of bytes
    out=fscanf(fid.s);  %scan the buffer
    x=strfind(out,'~'); %find out the start address of the start/end bytes
for each data value
%    fprintf('%d',x);
    if(isempty(x))  %if there exisits the desired identifiers
        fprintf('%s\n','Not found');
    else
        fprintf('%s\n','found');
       for i=1:2:(numel(x)-1) %loop around based on number of start bytes we
have
       %y=out(1,x(i):x(i+1));
       out(x(i))=' ';   %insert a space instead of the identifiers
       out(x(i+1))=' ';
       y=out(1,x(i):x(i+1));
       temp=str2double(y);   %extract the string and converted it to number
       if(rem(i,2)~=0)
            raw_data(iteration,j)=temp; %add it to the raw data value
            j=j+1;
       end
     %  temperature(iteration,1)=temp;   %add the value to the column array
       end
    end
```

```
   %temperature(iteration,:)=out(x(1));
   %data=horzcat(data,out); %keep on adding the data into a single character
array
   % out=[];
    b=0;
    %Calculate the voltage and current in the peltier loop
    V=raw_data(iteration,6)*Vref*VScale/1024;
    I_vout=raw_data(iteration,5)*Vref*IScale/1024;
    I=-36.067*I_vout^2 + 255.8*I_vout -423.16;
    %calculate the power consumed in the peltier in watts
    Power(iteration,1)=V*I;
    %update the power value on the GUI display
    set(handles.Power,'String',num2str(Power(iteration),5));
    if(iteration==str2double(no_iterations)) %loop for the number of
iterations defined by the user
       set(handles.Status,'String','Received Successfully');
       break;
    end
    iteration=iteration+1; %increment the counter after each successful
receive
  else           %if no data is available, wait for 10 seconds and check
again
      pause(10);
      if(fid.s.bytesAvailable()==0) %display an error message if no data
after 10s
          msgbox('No Data Received','TimeOut Error','Error');
          set(handles.Status,'String','Reception Unsuccessful');
          break;
      end
  end
end
fclose(fid.s);  %close the serial port
save ('Data.mat','raw_data','Power'); %save the required data variables

catch err4          %Catch error messages anywhere in the above code
   % display(err4.identifier);
    %Handle the error by updating the status
    if(strcmp(err4.identifier,'MATLAB:serial:fopen:opfailed'))
        set(handles.Status,'String','Serial Port Not Open');
    else if(strcmp(err4.identifier,'MATLAB:load:couldNotReadFile'))
        set(handles.Status,'String','serial.mat not found. Please open Serial
Port.');
        end
    end
end



%----------------------End of Data Acquisition Part----------------------%


% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider
value=get(hObject,'Value'); %get the value of the slider for a setpoint
%value=round(value);          %round the values to remove any decimals
%save('Data.mat','value');    %save the value of setpoint
%save the value of slider in the object property
setappdata(hObject,'slider1',value);
%update the value in the display
set(handles.sliderVal,'String',num2str(value,3));

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function sliderVal_Callback(hObject, eventdata, handles)
% hObject    handle to sliderVal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of sliderVal as text
%        str2double(get(hObject,'String')) returns contents of sliderVal as a
double
value=str2double(get(hObject,'String'));    %check for user entered value
if (isnumeric(value) && ...     %check if it is numeric
   value >= get(handles.slider1,'Min') && ...%check if it is within bounds of
the sliders
   value <= get(handles.slider1,'Max'))
   set(handles.slider1,'Value',value);  %update the slider position based on
user value
   %store the data in the application space of the display object
   setappdata(hObject,'sliderVal',get(hObject,'String'));
   setappdata(handles.slider1,'slider1',str2double(get(hObject,'String')));
   %save('Data.mat','value');    %save the value
end

% --- Executes during object creation, after setting all properties.
function sliderVal_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sliderVal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in setPoint.
function setPoint_Callback(hObject, eventdata, handles)
% hObject    handle to setPoint (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
instrreset;
clc;
%load the value of temperature desired from the workspace of slider1
value=getappdata(handles.slider1,'slider1');
try            %try-catch command for error handling
fid=load('serial.mat','s');
fopen(fid.s);   %open the serial port
value=value*10;
setpoint=num2str(value,3);
%fprintf(setpoint);
set(handles.Status,'String','Transmitting');
%transmit the command to update the setpoint
fprintf(fid.s,'w');
set(fid.s,'terminator','LF'); %configure the terminating character in the
communication
%transmit the setpoint
fprintf(fid.s,setpoint);
pause(1);
% check for the acknowledgement sent by the controllers
        if(fid.s.bytesavailable()>0)
            status=fscanf(fid.s);
            index=strfind(status,'OK');
            %msgbox(status,'Status','warn');
            if(isempty(index))
                set(handles.Status,'String','Data Transmission Failed');
            else
                set(handles.Status,'String','Data Transmitted Successfully');
            end
        else
            set(handles.Status,'String','Data Transmission Failed');
        end
fclose(fid.s);
catch err3      %error detection and handling  by the catch error statement

if(strcmp(err3.identifier,'MATLAB:serial:fopen:opfailed')||strcmp(err3.identi
fier,'MATLAB:load:couldNotReadFile'))
        set(handles.Status,'String','Serial Port Not Open');
    end
end

%--------------Update the activity of GUI--------------%
function Status_Callback(hObject, eventdata, handles)
% hObject    handle to Status (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Status as text
%        str2double(get(hObject,'String')) returns contents of Status as a
double
```

```
% --- Executes during object creation, after setting all properties.
function Status_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Status (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%-----------------------Serial Comm functions-------------------------%

%-----Get the serial port value-----%
function commSelect_Callback(hObject, eventdata, handles)
% hObject    handle to commSelect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of commSelect as text
%        str2double(get(hObject,'String')) returns contents of commSelect as
a double
com=get(hObject,'String');
%save the serial port value in the workspace of the Object
setappdata(handles.commSelect, 'comval', com);


% --- Executes during object creation, after setting all properties.
function commSelect_CreateFcn(hObject, eventdata, handles)
% hObject    handle to commSelect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in serialOpen - Open serial Port---%
function serialOpen_Callback(hObject, eventdata, handles)
% hObject    handle to serialOpen (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
instrreset; %resets the devices communicating on the serial port
clc;    %clear command window
try     %try-catch function for error handling
%load the com port value
COMPORT=getappdata(handles.commSelect,'comval');
s=serial(COMPORT);   %Set the communication port
```

```
set(s,'terminator','CR'); %configure the terminating character in the
communication
fopen(s);   %open the serial port
save('serial.mat','s');
set(handles.Status,'String','Serial Port Opened Successfully');
catch err1  %handle the errors and update the status message
    %display(err1.identifier);
    if (strcmp(err1.identifier,'MATLAB:serial:fopen:opfailed') ||...
        strcmp(err1.identifier,'MATLAB:serial:serial:invalidPORT') ||...
        strcmp(err1.identifier,'MATLAB:badfid_mx'))
        set(handles.Status,'String','Failed to open Serial Port. Enter Valid
COMPORT.');
    end
end

 % --- Executes on button press in serialClose-Close the serial Port ---%
function serialClose_Callback(hObject, eventdata, handles)
% hObject    handle to serialClose (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
try
serial=load('serial.mat','s');
fclose(serial.s);
delete(serial.s);
clear serial;
delete serial.mat;
set(handles.Status,'String','Serial Port Closed Successfully');
clear all;
catch err
    %display(err.identifier);
    if (strcmp(err.identifier,'MATLAB:load:couldNotReadFile'))
        set(handles.Status,'String','Serial.mat does not exist. Port is
already closed');
    end
end


%--------------------End of Serial Comm Functions----------------------%

%--------------------Set Destination Node Addrress---------------------%
%------Accept User input for MSBytes------%
function destAddMSB_Callback(hObject, eventdata, handles)
% hObject    handle to destAddMSB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of destAddMSB as text
%        str2double(get(hObject,'String')) returns contents of destAddMSB as
a double
%check for compatible input address
if(length(get(hObject,'String'))==8)
    %save the address MSB in the workspace of the Object
    setappdata(hObject,'highbytes',get(hObject,'String'));
else
    msgbox('Incompatible Input. Please enter 8 Hex Characters','Input
Error','Warn');
end
```

```
% --- Executes during object creation, after setting all properties.
function destAddMSB_CreateFcn(hObject, eventdata, handles)
% hObject    handle to destAddMSB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


%------Accept User input for LSBytes------%
function destAddLSB_Callback(hObject, eventdata, handles)
% hObject    handle to destAddLSB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of destAddLSB as text
%        str2double(get(hObject,'String')) returns contents of destAddLSB as
a double
%check for compatible input address
if(length(get(hObject,'String'))==8)
    %save the address LSB in the workspace of the Object
    setappdata(hObject,'lowbytes',get(hObject,'String'));
else
    msgbox('Incompatible Input. Please enter 8 Hex Characters','Input
Error','Warn');
end


% --- Executes during object creation, after setting all properties.
function destAddLSB_CreateFcn(hObject, eventdata, handles)
% hObject    handle to destAddLSB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%----Sets the user entered destination address----%
% --- Executes on button press in setDestAddr.
function setDestAddr_Callback(hObject, eventdata, handles)
% hObject    handle to setDestAddr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
instrreset; %resets the serial devices
```

```
clc;        %clear command line
%load the serial port object
try
fid=load('serial.mat','s');
fopen(fid.s);   %open the serial port
set(fid.s,'terminator','CR'); %configure the terminating character in the
communication
%form the required string to set the address in the Zigbee
msb=horzcat('ATDH',getappdata(handles.destAddMSB,'highbytes'));
lsb=horzcat('ATDL',getappdata(handles.destAddLSB,'lowbytes'));

fwrite(fid.s,'+++');
pause(1);
%fprintf(fscanf(fid.s));
if(fid.s.bytesavailable()>0)
    status=fscanf(fid.s);
    index=strfind(status,'OK');
    %msgbox(status,'Status','warn');
    if(isempty(index))
        set(handles.Status,'String','Failed to Enter Command Mode');
    else
        set(handles.Status,'String','Entering Command Mode');
        fprintf(fid.s,msb);
        pause(0.1);
        fprintf(fid.s,lsb);
        pause(0.1);
        fprintf(fid.s,'ATWR');
        pause(1);
        if(~isempty(strfind(fscanf(fid.s),'OK')))
            set(handles.Status,'String','Destination Address Changed
Successfully');
        end
    end
end
catch err

if(strcmp(err.identifier,'MATLAB:serial:fopen:opfailed')||strcmp(err.identifi
er,'MATLAB:load:couldNotReadFile'))
        set(handles.Status,'String','Serial Port Not Open. Open the Serial
Port.');
    end
end
%-------------------------Set Gain Parameters----------------------------%
function Kp_Callback(hObject, eventdata, handles)
% hObject    handle to Kp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Kp as text
%        str2double(get(hObject,'String')) returns contents of Kp as a double
val=get(hObject,'String');
index=strfind(val,'.');
if(isempty(index) && ~isnan(str2double(val))) %check for non integer input
    setappdata(hObject,'kp',str2double(val));
else
    msgbox('Incompatible Input. Please enter integer values.','Input
Error','Warn');
end
```

```
% --- Executes during object creation, after setting all properties.
function Kp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Kp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Ki_Callback(hObject, eventdata, handles)
% hObject    handle to Ki (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Ki as text
%        str2double(get(hObject,'String')) returns contents of Ki as a double
val=get(hObject,'String');
index=strfind(val,'.');
if(isempty(index) && ~isnan(str2double(val))) %check for non integer input
    setappdata(hObject,'ki',str2double(val));
else
    msgbox('Incompatible Input. Please enter integer values.','Input
Error','Warn');
end



% --- Executes during object creation, after setting all properties.
function Ki_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Ki (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in setGains.
function setGains_Callback(hObject, eventdata, handles)
% hObject    handle to setGains (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
instrreset;
clc;
try
```

```
fid=load('serial.mat','s');
fopen(fid.s);   %open the serial port

%load the Ki and Kp values saved in their respective object workspace
Kp=getappdata(handles.Kp,'kp');
Ki=getappdata(handles.Ki,'ki');

Kp=round(Kp);
Ki=round(Ki);
    set(handles.Status,'String','Transmitting');
    %send the command
    fprintf(fid.s,'g');
    set(fid.s,'terminator','LF'); %configure the terminating character in the
communication
    %send Kp value
    fprintf(fid.s,num2str(Kp));
    pause(0.1);
    %send Ki value
    fprintf(fid.s,num2str(Ki));
    pause(1);
    %check the acknowledgement sent by the controller
    if(fid.s.bytesavailable()>0)
        status=fscanf(fid.s);
        index=strfind(status,'OK');
        %msgbox(status,'Status','warn');
        if(isempty(index))
            set(handles.Status,'String','Failed to set Parameters');
        else
            set(handles.Status,'String','Parameters Set');
        end
    else
        set(handles.Status,'String','Failed to Set Parameters');
    end
fclose(fid.s);
catch err

if(strcmp(err.identifier,'MATLAB:serial:fopen:opfailed')||strcmp(err.identifi
er,'MATLAB:load:couldNotReadFile'))
        set(handles.Status,'String','Serial Port Not Open. Open the Serial
Port.');
    end
end
%-------------------------End of gain settings-------------------------%


%-----------------------------Plot Functions-----------------------------%
% --- Executes on button press in plot1.
function plot1_Callback(hObject, eventdata, handles)
% hObject    handle to plot1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%load all the data variables
data= load ('Data.mat','raw_data');

iterations=getappdata(handles.iterationVal,'itrVal');
iterations=str2double(iterations);
interval=getappdata(handles.readInterval,'interVal');
```

```
interval=str2double(interval)/1000;
temperature1=data.raw_data(1:iterations,1)/10;
endTime=interval*iterations;
%calculate the time axis
time=0:interval:endTime-1;
figure(1);
plot(time,temperature1);
title('Transient Response of the Peltier Module');
xlabel('Time(seconds)');
ylabel('Peltier Outlet Temperature(C)');

% --- Executes on button press in plot2.
function plot2_Callback(hObject, eventdata, handles)
% hObject    handle to plot2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
data= load ('Data.mat','raw_data');
iterations=getappdata(handles.iterationVal,'itrVal');
iterations=str2double(iterations);
interval=getappdata(handles.readInterval,'interVal');
interval=str2double(interval)/1000;
temperature2=data.raw_data(1:iterations,2)/10;
endTime=interval*iterations;
time=0:interval:endTime-1;

figure(2);
plot(time,temperature2);
title('Transient Response of the Heat Exchanger');
xlabel('Time(seconds)');
ylabel('Condensor Outlet Temperature(C)');


% --- Executes on button press in plotPower.
function plotPower_Callback(hObject, eventdata, handles)
% hObject    handle to plotPower (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
data= load ('Data.mat','Power');
iterations=getappdata(handles.iterationVal,'itrVal');
iterations=str2double(iterations);

interval=getappdata(handles.readInterval,'interVal');
interval=str2double(interval)/1000;
endTime=interval*iterations;
time=0:interval:endTime-1;
figure(3);
plot(time,data.Power);
title('Power consumed by the Peltier Module');
xlabel('Time(seconds)');
ylabel('Power(Watts)');

%-------------------------End of Plot Functions--------------------------%

%---------------------------Menu Bar Functions---------------------------%

function file_Callback(hObject, eventdata, handles)
% hObject    handle to file (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% -----save the enter values to a file -----%
function saveSettings_Callback(hObject, eventdata, handles)
% hObject    handle to saveSettings (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA
iterations=getappdata(handles.iterationVal,'itrVal');
interval=getappdata(handles.readInterval,'interVal');
temperature=getappdata(handles.sliderVal,'sliderVal');
kp=getappdata(handles.Kp,'kp');
ki=getappdata(handles.Ki,'ki');
save('Parameters.mat','iterations','interval','temperature','kp','ki');
set(handles.Status,'String','Settings Saved');

% ----- Load the saved values from file -----%
function loadSettings_Callback(hObject, eventdata, handles)
% hObject    handle to loadSettings (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load Parameters.mat;
setappdata(handles.iterationVal,'itrVal',iterations);
set(handles.iterationVal,'String',iterations);
setappdata(handles.readInterval,'interVal',interval);
set(handles.readInterval,'String',interval);
setappdata(handles.slider1,'slider1',temperature);
set(handles.slider1,'Value',str2double(temperature));
setappdata(handles.sliderVal,'sliderVal',temperature);
set(handles.sliderVal,'String',temperature);
setappdata(handles.Kp,'kp',kp);
set(handles.Kp,'String',num2str(kp));
setappdata(handles.Ki,'ki',ki);
set(handles.Ki,'String',num2str(ki));
set(handles.Status,'String','Settings Loaded');

% --------------------------------------------------------------------
function help_Callback(hObject, eventdata, handles)
% hObject    handle to help (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function Topics_Callback(hObject, eventdata, handles)
% hObject    handle to Topics (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
open('Troubleshooting.pdf');

% --------------------------------------------------------------------
function about_Callback(hObject, eventdata, handles)
% hObject    handle to about (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
text = fileread('About.txt');
```

```
msgbox(text,'About','help');
% --------------------------------------------------------------------
%----------------------End of Menu Bar Functions-------------------------%

function Power_Callback(hObject, eventdata, handles)
% hObject    handle to Power (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Power as text
%        str2double(get(hObject,'String')) returns contents of Power as a
double


% --- Executes during object creation, after setting all properties.
function Power_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Power (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Appendix D: Modelica and Matlab code

```
model Pump_Test
parameter Real QuadF=-0.0000168;
parameter Real LinearF=0.0075;
parameter Real ConstantF=0.0043;
Modelica.Fluid.Sources.FixedBoundary Source(
   redeclare package Medium =
      Modelica.Media.Water.ConstantPropertyLiquidWater,
   nPorts=1,
   p=100000,
   T=293.15)
   annotation (Placement(transformation(extent={{-62,-30},{-42,-10}})));
 Modelica.Fluid.Sources.FixedBoundary Sink(
   redeclare package Medium =
      Modelica.Media.Water.ConstantPropertyLiquidWater,
   use_p=true,
   use_T=true,
   nPorts=1,
   p=100000,
   T=293.15)
   annotation (Placement(transformation(extent={{-10,-10},{10,10}},
      rotation=180,
      origin={68,-20})));
 inner Modelica.Fluid.System system(p_ambient=100000, T_ambient=293.15)
   annotation (Placement(transformation(extent={{40,0},{60,20}})));
 Modelica.Mechanics.Rotational.Sources.ConstantSpeed constantSpeed(w_fixed(
      displayUnit="rad/s") = 154)
         annotation (Placement(transformation(extent={{-40,0},{-20,20}})));
 Modelica.Blocks.Sources.Constant const(k=1)
   annotation (Placement(transformation(extent={{0,0},{20,20}})));
 Modelica.Blocks.Sources.Ramp ramp(
   duration=10,
   height=.9,
   offset=.1,
   startTime=5)
          annotation (Placement(transformation(extent={{36,74},{56,94}})));
 PumpModel pumpModel(
   redeclare package Medium =
      Modelica.Media.Water.ConstantPropertyLiquidWater,
   redeclare function flowCharacteristic = Building_HVAC.QuadFlow (
      A=QuadF,
      B=LinearF,
      C=ConstantF),
   use_powerCharacteristic=true,
   redeclare function powerCharacteristic = Building_HVAC.QuadPower (
      A=CubicP,
      B=QuadP,
      C=LinearP,
      D=ConstantP),
   N_nominal(displayUnit="1/min") = 725)
   annotation (Placement(transformation(extent={{-20,-30},{0,-10}})));
 Modelica.Fluid.Valves.ValveLinear valveLinear(
   redeclare package Medium =
      Modelica.Media.Water.ConstantPropertyLiquidWater,
   m_flow_nominal=1,
   dp_nominal=100000)
   annotation (Placement(transformation(extent={{20,-30},{40,-10}})));
equation
 connect(Source.ports[1], pumpModel.port_a) annotation (Line(
    points={{-42,-20},{-20,-20}},
    color={0,127,255},
    smooth=Smooth.None));
 connect(constantSpeed.flange, pumpModel.shaft) annotation (Line(
    points={{-20,10},{-10,10},{-10,-10}},
    color={0,0,0},
    smooth=Smooth.None));
 connect(pumpModel.port_b, valveLinear.port_a) annotation (Line(
```

```
    points={{0,-20},{20,-20}},
    color={0,127,255},
    smooth=Smooth.None));
 connect(valveLinear.port_b, Sink.ports[1]) annotation (Line(
    points={{40,-20},{58,-20}},
    color={0,127,255},
    smooth=Smooth.None));
 connect(const.y, valveLinear.opening) annotation (Line(
    points={{21,10},{30,10},{30,-12}},
    color={0,0,127},
    smooth=Smooth.None));
 annotation (                    Diagram(graphics),
   experiment(StartTime=1, StopTime=20),
   __Dymola_experimentSetupOutput);
end Pump_Test;
```

Matlab code:

```
clear all
dymosim([1 20,0,500,1e-4,8]);
D=dymload;
%dp=dymget(D,'Pump.dp_pump');
%dp=dp/(8.91*10^5); %Pressure drop in meters
%Vflow=dymget(D,'Pump.V_flow');
%Vflow=Vflow/(1.56*10^(-4)) %Flow rate in liters per minute
%k=dymget(D,'const.k');
%t=dymget(D,'Time');
%plot(Vflow,dp);
%plot(t,dp)
speed=[76 86 95 100 113 121 127 135 144 154];
for k=1:length(speed)
[p xo pnames xonames]=loaddsin('dsin.txt');
pout=setParameterByName(pnames, p,'constantSpeed.w_fixed',speed(k));
dymosim([1 20,0,500,1e-4,8],xo,pout);
D=dymload;
dp(:,k)=dymget(D,'pumpModel.dp_pump');
dp(:,k)=dp(:,k)./100;
Vflow(:,k)=dymget(D,'pumpModel.V_flow');
Vflow(:,k)=Vflow(:,k).*1000*3600;
Power(:,k)=dymget(D,'pumpModel.W_total');
eff(k)=dp(k)*Vflow(k)/Power(k);
end
 %Vflow=-Vflow*1000*3600;
%k=dymget(D,'const.k');
t=dymget(D,'Time');
plot (Vflow,dp)
%------different m-file----%
function [f]= myfun()
[x fval]=fmincon(@myfun,0,[],[],[],[],0,300)
end
%------different m-file----%
function f=myfun(x)
% x = flow rate in LPM
% power =(0.000002*x(1)^2-0.0089*x(1) + 12.248
f=(0.0000168*x^2-0.0075*x-0.0043);
end
```

Appendix E: Troubleshooting the GUI:

COMPORT – Enter the address of the Comport after looking it up in the device manager.

Possible Errors:

1. Invalid ComPort. Enter a valid Comport

If no COMPORT value is entered or a non-existing COMPORT is entered, you may get this error message.

Look up the correct name of the port from the 'Device Manager' under properties of 'Computer'. Try to open the Serial Port after the change.

Destination Address – Enter the address of the wireless node you want the data from.

Possible Errors:

1. 'Enter 8 Hex Characters of the MSB':

If you enter less or more than 8 characters, this error will appear in a message box. Revise the address and enter it again in the edit box.

Number of Reads and Read Interval – The user enters the number of data point he wants and the time interval between consecutive data points.

Possible Errors:

1. Entering non integer values:

Entering floating point values here would cause an error message to pop up. Rectify the entered number.

2. Entering zero:

Entering Zero will give an error message and would default the number of reads to 20.

3. Entering alphabets in the edit box:

If the input is alphanumeric or alphabetic, an error message will pop up. Please rectify the input and retry.

Gains Kp and Ki – The user enter the gains for the PI loop controlling the Peltier heater.

Possible Errors:

1. Floating point of alphanumeric or alphabetic input by the user:

If such values are entered it will cause the GUI to throw an error message pop-up. Rectify the values based on the instruction and retry.

Default Settings:

The default settings have been configured for the following:

1. Number of iterations is 20 if invalid value is entered and used
2. The read interval is set to 1000 ms if invalid value is entered and used.
3. Temperature is set to 24˚C.
4. Before using the Set Gain Function in the GUI, one must take care to enter valid values. If valid values are not used, it may lead to unpredictable behavior of the system.

BIBLIOGRAPHY

[1] REMI ALLET, "*Development of models for designing industrial energy technologies related to cold production and storage*" Goteborg, Sweden 2011
http://publications.lib.chalmers.se/records/fulltext/145874.pdf

[2] Allan, John and Nekimken Kyle and Weills Spencer, 2009 "*Pump Efficiency Solutions*", California.

[3] Zhang, He, Xiaohua Xia, and Jiangfeng Zhang. "*Optimal Sizing and Operation of Pumping Systems to Achieve Energy Efficiency and Load Shifting*." *Electric Power Systems Research* 86, no. 0 (May 2012): 41–50.

[4] Ma, Y., Borrelli, F., Hencey, B., Coffey, B., Bengea, S., & Haves, P. (2010). Model predictive control for the operation of building cooling systems. *American Control Conference (ACC), 2010* (pp. 5106 – 5111).

[5] Faludi, R. (2010). *Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing* (1st ed.). O'Reilly Media.

[6] Hans Olsson, Martin Otter, Sven Mattsson, Hilding Elmqvist, "*Balanced Models in Modelica 3.0 for Increased Model Quality*" March 4th, 2008
https://modelica.org/events/modelica2008/Proceedings/sessions/session1a3.pdf

[7 KSB, Volute Casing Pump Offer Curves, KSB industry, 2007

[8] "Electronics Cooling Methods in Industry http://www.pathways.cu.edu.eg/ec/Text-PDF/Part%20C-17.pdf

[9] TIAX, LLC. Energy impact of commercial building controls and performance diagnostics: Market characterization, energy impact of building faults and energy savings potential. Report No. D0180, Prepared for the DOE Building Technologies Program. 2005.

[10] CBECS 2003. Commercial Buildings Energy Consumption Survey commercial energy uses and costs. Energy Information Administration
[12] https://www.modelica.org/documents/ModelicaTutorial14.pdf

[13] "*Buildings and their Impact on the Environment"*: A Statistical Summary, 2009
http://www.epa.gov/greenbuilding/pubs/gbstats.pdf

[14] Haves, P., Hencey, B., Borrell, F., Elliot, J., Ma, Y., Coffey, B., Bengea, S., et al. (2010). *Model Predictive Control of HVAC Systems: Implementation and Testing at the University of California, Merced*. Retrieved from http://www.osti.gov/bridge/servlets/purl/988177-RRFcmS/

[15] Yawut Cholatip and Sathapath Kilaso, 2011, "*A Wireless Sesnsor Network for Weather and Disaster Alarm Systems*" Singapore, IACSIT Press.

[16] Fetyan Khaled, and Younes M, and Helal M, 2007 "*Energy Saving of Adjustable Speed Pump Stations in Egypt*" Cairo, Egypt

[17] http://newscenter.lbl.gov/feature-stories/2009/06/02/working-toward-the-very-low-energy-consumption-building-of-the-future/

[18] B. Erpelding, "Real Efficiencies of Central Plants," HPAC Engineering, May 2007

[20] Haves, P., Hencey, B., Borrell, F., Elliot, J., Ma, Y., Coffey, B., Bengea, S., et al. (2010). *Model Predictive Control of HVAC Systems: Implementation and Testing at the University of California, Merced*. Retrieved from http://www.osti.gov/bridge/servlets/purl/988177-RRFcmS/

[21] Martin, David (2010), "*Functional Platform for Rapid Development of Sustainable Building Systems"* a master of engineering thesis, Cornell University.

[22] Flinton,Dave, ITT Industrial & BioPharm Group (2006) "Optimizing Pump Efficiency"

[23] http://www.cee1.org/cee/mtg/09-06_ppt/pumps-2.pdf

[24] www.Avrfreaks.net

[25] www.arduino.cc

[26] http://www.dymola.com

[27] http://www.mathworks.com

[28] Atemga 328p datasheet: http://www.atmel.com/Images/8271S.pdf

[29] http://techteach.no/publications/articles/zn_closed_loop_method/zn_closed_loop_method.pdf

[30] http://www.embedded.arch.ethz.ch/xbee-setup.pdf

[31] B. Coeffey, F. Haghighat, E. Morofsky, and E. Kutrowski. "A software framework for model predictive control with GenOpt," *Energy and Buildings*, vol. 42, pp. 1084-1092, 2010.

[32] S. Liu and G.P. Henze. "Calibration of buildings models for supervisory control of commercial buildings," in *Proceedings of building simulation 2005*, Montreal, Canada, pp. 641-648, Aug. 2005.

[34] Newsham, Guy R., Sandra Mancini, and Benjamin J. Birt. "Do LEED-certified Buildings Save Energy? Yes, But..." Energy and Buildings 41.8 (2009) : 897-905. Web. 13 Mar 2010.

[35] Jain, N., Li, B., Keir, M., Hencey, B., & Alleyne, A. (2010). Decentralized Feedback Structures of a Vapor Compression Cycle System. *Control Systems Technology*, *IEEE Transactions on*, *18*(1), 185 –193. doi:10.1109/TCST.2008.2010500

[36] R. Shah, B. Rasmussen, and A. Alleyne, "Application of multivariable adaptive control to automotive air conditioning systems," Int. J. Adapt. Control Signal Process., vol. 18, no. 2, pp. 199–221, Mar. 2004.

[37] M. C. Keir, "Dynamic modeling, control, and fault detection in vapor compression systems," M.S. thesis, Dept. Mech. Eng., Univ. Illinois Urbana–Champaign, Urbana, IL, 2006.

[38] http://www.fypower.org/pdf/BPG_hotels.pdf

[39] http://buildingsdatabook.eren.doe.gov/TableView.aspx?table=1.1.3